NASA Contractor Report 3651

Measurement of Fault Latency in a Digital Avionic Mini Processor

Part II

N83-17097

John G. McGough and Fred L. Swern

CONTRACT NAS1-15946
JANUARY 1983

NASA

NASA Contractor Report 3651

Measurement of Fault Latency in a Digital Avionic Mini Processor

Part II

John G. McGough and Fred L. Swern
Bendix Corporation
Teterboro, New Jersey

Prepared for
Langley Research Center
under Contract NAS1-15946

NASA

National Aeronautics
and Space Administration

Scientific and Technical
Information Branch

1983

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1.0 INTRODUCTION

1.1 Background

This study is a follow-on of an earlier study entitled:

"Measurement of Fault Latency in a Digital Avionic Mini Processor" (ref. 1).

To place the present study in perspective we include a brief summary of the results of the earlier study:

Summary of Earlier Study

- A gate level emulation of the Bendix BDX-930 digital computer was developed for the purpose of analyzing failure modes and effects in digital systems. The run time of the emulator was 7000 times slower than the BDX-930 when hosted on a VAX 11/780.

- Six software programs were emulated and faults were injected at both the gate-level and pin-level (i.e., component-level). The resultant computed outputs were compared with those of a non-faulted computer executing the same program. A fault was considered detected when these outputs differed. The results showed that:

  .. Most detected faults are detected in the first repetition. Subsequent repetitions do not appreciably increase the proportion of detected faults.

  .. A large proportion of faults remained undetected after as many as 8 repetitions of the program, e.g., 60% at the gate-level.

  .. Component-level faults are easier to detect than gate-level faults. For example, after 8 repetitions, the proportion of undetected faults were

GATE-LEVEL	COMPONENT-LEVEL
61.7%	35.5%
58.2%	28%
59.5%	32.3%

for the program FETSTO, FIB and ADDSUB, respectively. To corroborate the findings of a pilot study (ref. 2) the instruction repertoire of the BDX-930 was limited to the following instructions:

Load
Store
Add
Subtract
Branch
Transfer
Clear

- The results of the study corroborated the findings of the pilot study of (ref. 2). This was surprising considering that the pilot study used an emulation of a very simple processor. As an illustration, the pilot study indicated that, after 8 repetitions, the proportion of undetected faults were

  64.4%
  53.7%
  44.9%

  for FETSTO, FIB, ADDSUB, respectively.

- The Urn Model, for forecasting fault latency, produced distributions that were in close agreement with the empirical distributions.

- A self-test program of 2000 executable instructions was expressly designed for the study. The designer was given the single requirement that fault coverage should be at least 95%. The resultant test consisted of 241 separate subtests for the purpose of exercising the entire instruction set of the BDX-930.

  The results indicated that there is a significant difference in coverage of gate-level versus component-level faults. For example,

gate-level coverage	=	86.5%
component-level coverage	=	97.9%

- Only 48% of all detected faults were detected by a subtest. The remaining detected faults were detected because the first subtest was not computed.

- Most of the subtests were redundant, i.e., only 46 of 241 subtests actually detected a fault.

- 62% of all detected faults were detected by the first 23 subtests.

- A large proportion of "don't care" (i.e., indistinguishable) faults were injected: (i.e., 23.7%). These proved to be exceedingly difficult to identify.

- The micromemory prom contained the largest proportion of undetected faults.

To conserve space this report omits certain details which were contained in the earlier report, notably in the areas of statistical analyses and descriptions of the emulator. However, the present text is self-contained; whenever comparisons are necessary the pertinent data from the earlier report is duplicated.

1.2 Objectives of the Study

The poor coverage of comparison-monitoring, which the earlier study demonstrated, could have been due to the limited repertoire of the instruction set used. As a consequence, it was decided to reprogram SERCOM, LINCON and QUAD but this time expanding the instruction set to capitalize on the full power of the BDX-930. As a final demonstration of failure coverage an extensive, 3-axis, high performance flight control computation was added.

As the summary of the earlier study indicates, failure detection coverage of the target self-test program was a disappointing 86.5% for gate-level faults. As a consequence, Bendix conducted a development program (independent of the present contract) for the purpose of upgrading the self-test to approximately 95% coverage while minimizing the number of instructions and run-time. The initial self-test was used as a baseline. The successive self-test programs and their resultant coverages are described in Section 7.0.

1.3 Foreward

The use of trade names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied by the National Aeronautics and Space Administration.

2.0 SUMMARY

1. A gate-level emulation of the Bendix BDX-930 digital computer was used to perform fault injection experiments to determine a program's ability to detect faults. The emulator was hosted on a VAX-11/780. The resultant run-time was 7000 times slower than the BDX-930.

2. Four software programs were emulated and faults were injected at both the gate-level and pin-level (i.e., component-level). The resultant computed outputs were compared with those of a non-faulted computer executing the same program. A fault was considered detected when their outputs differed.

3. The present study was a follow-on to a previous study in which the programs were limited to a simple instruction set. The four programs of the present study used the full power of the BDX-930 instruction set. One of the four programs included a 3-axis, high performance flight control system of approximately 2200 executable instructions. The objective was to coroborate the conclusions of the previous study.

4. The results corroborated those of the previous study. In particular:

   - Most detected faults are detected in the first repetition of a program. Subsequent repetitions do not appreciably increase the proportion of detected faults.

   - Short programs have a tendency to benefit more from subsequent repetitions than lengthier programs.

   - A large proportion of gate-level faults remained undetected after as many as 8 repetitions of the program. For example, in the flight control computation, 21% of distinguishable (i.e., "CARE") faults remained undetected after 8 repetitions.

   - Pin-level faults are easier to detect than gate-level faults.

5. A self-test program should be designed to capitalize on the hardware mechanization of the CPU. A self-test designed to exercise instructions without regard for the hardware mechanization tends to be inefficient in real-time and memory.

6. The Urn Model can characterize fault latency distributions. It is doubtful, however, that the Urn Model parameters can be predicted on the basis of a program's length and instruction mix.

3.0 FAULT MODELLING AND SELECTION

3.1 Fault Model

At the present time there is little or no data available regarding either the mode or frequency of failures of MSI and LSI devices. Despite this deficiency of data, failure modes and effects analyses are regularly performed for avionics and flight control systems. The conventional approach is to assume a set of failure modes for each device. These are usually restricted to faults at single pins although, occasionally, multiple faults may be considered. In most cases the failure rate of a device is assumed to be equally distributed over the pins or over the set of postulated failure modes. Except for special devices, faults are assumed to be static, being either S-a-0 or S-a-1.

The point to be made here is that failure modes and their rate of occurrence are necessarily conjectural and the credibility of the present study suffers no less from this deficiency of data than the conventional analysis. The authors emphasize that the emulation approach does not solve this problem.

In the present study the following assumptions are made regarding failure modes:

- Every device can be represented, from the standpoint of performance and failure modes, by the manufacturer-supplied, gate-level equivalent circuit.
- Every fault can be represented as either S-a-0 or S-a-1 fault at a gate node.
- The failure rate of the device is equally distributed over the gates of the equivalent circuit.
- The failure rate of a gate is equally distributed over the nodes of the gate.
- S-a-0 and S-a-1 faults are equally likely.
- Memory faults are exclusively faults of single bits.
- A memory fault is the complement of its non-faulted state.

Faults are injected into all devices except the main memory. In the case of the microprogram memory, which is emulated at the functional level, faults are injected into the memory cells where they remain active for the duration of the test. Faults are injected at an input or output gate node, and also remain active for the duration of the test. When a fault is injected at an output node it is allowed to propagate to all nodes and devices that are physically connected to the failed node. When a fault is injected at an input node it does not propagate back to the driving node. This strategy provides a wider variety of failure modes than would otherwise be possible if propagation were allowed. The fault model, although conjectural at the present time, can be updated as fault data becomes available. The proposed model provides a simple, automatic and consistent method of generating faults. The resultant fault set includes a rich assortment of static and dynamic (i.e., data-dependent) faults.

3.2 Method of Selecting Faults

The method of selecting faults is implicit in the fault model. Explicitly,

- Each device is assigned a failure rate.
- The failure rate is equally distributed over the gates of the gate-level representation.
- The failure rate of each gate is equally distributed over the nodes of the gate.
- The failure rate of each node is equally distributed over S-a-0 and S-a-1 faults.
- As a result of this procedure, each S-a-0 and S-a-1 fault is assigned a probability of occurrence proportional to the prescribed failure rate. The resultant fault set is then randomly sampled with each fault weighted by its probability of occurrence. It is noted that, according to this procedure, faults in devices with high failure rates will be selected more frequently than faults in devices with lower failure rates.

The above procedure does not distinguish between gate-level and component (i.e.,pin)-level faults except by probability of occurrence; the method automatically assigns failure rates to pins. However, a different selection procedure was employed for component-level faults. For these faults is was assumed that:

- The failure rate of each device is equally distributed over the pins.

While this assumption violates the prescribed fault model it is consistent with the conventional method of estimating fault detection coverage by simulating faults in actual hardware. As a consequence, all component-level detection estimates obtained in the study are estimates that would be obtained by proponents of this approach.

4.0 DESCRIPTION OF EXPERIMENTS

4.1 Definition of Failure Detection

In the present study, fault coverage and latency estimates are obtained by employing two conventional techniques of failure detection: comparison-monitoring and self-test.

In comparison-monitoring a set of computed variables is compared with a corresponding set computed in another processor. If it is arranged that both processors operate on identical inputs and are closely synchronized, then any difference in a computed variable signifies that one of the processors has failed. In practice each processor executes an algorithm which compares the appropriate variables and signals a discrepancy when such exists. In the present study this algorithm was omitted; a fault is considered to be detected if a difference between corresponding variables exists irrespective of the ability of either processor to recognize the difference or signal the discrepancy. Thus, the fault coverage obtained from the study is somewhat more optimistic than would be obtained in practice.

In self-test, on the other hand, each component of the processor is exercised by a set of computations designed specifically to test that component. The results of each computational set are compared with pre-stored values and any difference signifies that the fault was detected. In practice, and in the study, the processor increments a register after the successful completion of each test and before proceeding to the next test. If the test is not successful the program exits. After an interval of time equal to the maximum time to complete the program, the contents of the counter are decoded. If the value exactly equals that total number of tests, the fault was not detected. Otherwise the fault was detected.

It is emphasized that "failure detection", as it is used in the present study, means almost exactly what it means in an actual airborne avionic system. This is in marked contrast to the commonly employed alternate approach of assuming that a failure is detected whenever the effect to the failure reaches an accessible bus or register, even though the program may not be interrogating these devices at that time.

In the following paragraphs a description is given of the actual computations involved in the experiment with particular emphasis on the explicit definition of "failure detection" in each instance.

4.2 Definition of Failure Detection Coverage

We assume that a test procedure is given for detecting failures of a component, C. Each failure mode of C will require a non-zero time for detection. By considering all failures of C and all combinations of inputs and internal states of C, we obtain in principle, if not in practice, a probability density function for time-to-detect, which is measured from the onset of the failure to the time of detection.

Denoting this density by pdf (τ) where

τ = time-to-detect = latency time

we define

Test Coverage

$$1)\quad 1 - \alpha(\tau) = \int_0^{\tau} \mathrm{pdf}(X)dX$$

= probability of detecting a failure of C in the interval $0 \leqq t \leqq \tau$.

Observe that, according to this definition, test coverage is a function of latency time. The definition can be extended to all devices of the computer as follows:

Subdivide the computer into mutually exclusive components $C_1, C_2, \ldots, C_k$ with failure rates $\lambda_1, \lambda_2, \ldots, \lambda_k$, and test coverages $1 - \alpha_1(\tau)$, $1 - \alpha_2(\tau), \ldots, 1 - \alpha_k(\tau)$, respectively.

Set $\quad \mathrm{pdf}_i(\tau)$ = probability density for time-to-detect failures of C_i, $i = 1, 2, \ldots, k$.

Then the pdf for all failures of the computer is

$$2)\quad \mathrm{pdf}(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda}\, \mathrm{pdf}_i(\tau)$$

where $\quad \lambda = \lambda_1 + \lambda_2 + \ldots. + \lambda_k$.

Test coverage of the whole computer is then

$$3)\quad 1 - \alpha(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda} (1 - \alpha_i(\tau)).$$

The method of selecting faults, described in Section 3, is consistent with this definition.

From (3) we obtain.

$$4)\quad \alpha(\tau) = \sum_{i=1}^{i=k} \frac{\lambda i}{\lambda} \alpha_i(\tau), \text{ as expected.}$$

One of the objectives of the present study is to obtain estimates of the probability density function, pdf(τ). These estimates are presented in Section 5.

4.3 Indistinguishable Faults and Effects on Coverage

During the development of the emulator it became apparent that a significant proportion of components had no affect whatsoever on the digital process. For the most part, these components are associated with unused pins, e.g., a complementary output of a flip-flop. However, there are other components whose lack of effect are not as obvious as, for example, a component that only affects the process when it is faulted. Certain micromemory bits are in this category. In order to distinguish between these categories of faults we are lead to the following informal definitions:

A fault that has no affect on the computational process is <u>indistinguishable</u>. All other faults are <u>distinguishable</u>.

We note that a distinguishable fault has the property that there exists a software program the output of which differs from that of the same program executed by an identical but non-faulted processor.

Effects on Coverage

The presence of indistinguishable faults can lead to erroneous and misleading estimates of coverage. In theory, indistinguishable faults should be disqualified from the emulation or from the fault selection process. This is consistent with the definition of coverage which implicitly assumes that faults are distinguishable. Unfortunately, in order to disqualify indistinguishable faults from the emulation or from the fault selection process they must be first identified and this is a non-trivial task because of the large number of possible faults. The approach taken in this study was to select faults irrespective of their distinguishability properties and analyze only those faults that were undetected by Self-Test. The proportion of indistinguishable faults from this set was then used as an estimate over all faults.

We now indicate, briefly, how indistinguishable faults affect coverage.

If

γ = proportion of components yielding indistinguishable faults

and

$1 - \alpha$ = coverage of distinguishable faults.

then

$1 - \alpha$ = desired coverage

and

5) $(1 - \alpha)(1 - \gamma)$ = coverage when indistinguishable faults are counted as undetected. We note, incidentally that

6) $(1 - \alpha)(1 - \gamma) + \gamma$ = coverage when indistinguishable faults are counted as detected.

The estimate of (5) will be obtained if indistinguishable faults are not disqualified. Then, coverage estimates will be in error by the factor, $1-\gamma$.

In the more general case it may be more convenient to estimate the proportion of indistinguishable faults by partition since the affect on coverage is a function of the relative failure rate of the partition.

Let λ_i = failure rate of Partition #i, i = 1, 2, ..., 6.

λ_i = proportion of indistinguishable faults in Partition #i.

$1 - \alpha_i$ = coverage of distinguishable faults in Partition #i.

$\lambda = \lambda_1 + \lambda_2 + \ldots + \lambda_6$ = total failure rate.

From the previous section, if all faults are distinguishable then coverage is given by

7) $$1 - \alpha = \sum_{i=1}^{6} \frac{\lambda i}{\lambda} (1 - \alpha_i)$$

If, however, indistinguishable faults are counted as undetected then the coverage actually obtained is

$$8)\quad 1 - \alpha = \sum_{i=1}^{6} \frac{\lambda i}{\lambda} (1 - \alpha_i)(1 - \gamma_i).$$

We note that, if indistinguishable faults are disqualified, the true coverage is

$$9)\quad 1 - \alpha = \frac{\sum_{i=1}^{6} \lambda_i (1 - \alpha_i)(1 - \gamma_i)}{\sum_{i=1}^{6} (1 - \gamma_i) \lambda_i} .$$

From (8) it can be seen that the required accuracy of an estimate of γ_i depends upon the relative failure rate, λ_i/λ. If λ_i is sufficiently small then the effect of an inaccurate estimate of γ_i is negligible.

4.4 Objectives of Experiments

Most airborne systems, present and projected, employ comparison-monitoring, self-test or a combination of both to achieve the requisite detection and isolation capability. One of the problems of fault detection, by either method, is that a fault may not manifest itself at either a comparison-monitored variable or at an accessible output of self-test until the faulted component is suitably exercised. As a consequence, faults can remain latent for long periods of time. This is the significance of latency time, τ, in the definition of test coverage of Section 4.2.

One of the objectives of the experiments is to estimate τ for the test programs described in Section 4.5. Using comparison-monitoring the probability distribution of τ will be estimated for each of the four programs and the interdependence of these distributions and the number and type of instructions will be ascertained.

4.5 Experiments

The fault injection experiments were conducted using four programs, each of which was coded in the assembly language of the BDX-930.

In the following descriptions only the set of computations labelled "compute" were performed by the target BDX-930 CPU; all other computations, selections, comparisons, etc. were performed by the emulation host computer Executive. Needless to say, there were no failures in these latter computations.

When the non-failed processor completed a computation* and before the start of the next computation the Executive recomputed all initializing variables and stored them in the appropriate locations of the scratchpad memory.

In the parallel mode of operation, when 32 computers are simultaneously being emulated, the initializing variables are stored simultaneously in the 32 copies of the scratchpad memories.

4.5.1 Search and Compute (SERCOM)

a. Procedure

T0) Select 8 sets of integers, (A_k, B_k, C_k), at random, each component from the interval

$$0 \leqq x \leqq 20.$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the (A_k, B_k, C_k) in successive locations of memory.

T3) Compute and store in successive locations of memory

$$\left.\begin{aligned} S_{1k} &= B_k + C_k \\ S_{2k} &= B_k \end{aligned}\right\} \quad \text{if } B_k \leqq A_k$$

$$\left.\begin{aligned} S_{1k} &= B_k + C_k \\ S_{2k} &= B_k - C_k \end{aligned}\right\} \quad \text{if } A_k < B_k \text{ and } C_k \leqq A_k$$

* In the parallel mode of operation one of the emulated processors is non-faulted and, as a consequence, the end of its computation cycle can be determined from its program counter.

$$\left.\begin{aligned} S_{1k} &= B_k - C_k \\ S_{2k} &= B_k \times C_k \end{aligned}\right\} \qquad \text{if } A_k < B_k \text{ and } A_k < C_k.$$

T4) When the non-failed processor completes its last instruction compare S_{1k}, S_{2k} term by term, in both the non-failed and failed processors. If S_{1k} or S_{2k} is the first variable to miscompare set L = K. If all S_{1k}, S_{2k} compare set L = 0 (L = Latency Period).

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD/STORE	11
STACK OPS	4
ADD/SUBTRACT	4
MULTIPLY	1
BRANCH	12
TRANSFER	8
MISCELLANEOUS	4
	44

4.5.2 Linear Convergence (LINCON)

a. Procedure

T0) Select the following integers from the indicated intervals:

M_o,	$-8 \leqq M_o \leqq 8$
Y_o,	$-2^{14} + 1 \leqq Y_o \leqq 2^{14} - 1$
$X_1, X_2, \ldots., X_8$,	$0 \leqq X_k \leqq 2^{11}$.

Assume that $X_1 < X_2 < \ldots. < X_8$.

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store M_o, Y_o, X_1, X_2, ..., X_8, in successive locations of memory.

T3) Compute M_k, Y_k, for K = 1, 2, ..., 8 as specified in the flow diagram of Figure 1, and store in successive locations of memory.

T4) When the non-failed processor completes its last instruction compare M_1, Y_1, ..., M_8, Y_8, term by term, in both the non-failed and failed processors. If M_k or Y_k is the first variable to miscompare set L=K. If all M_k, Y_k compare set L=0.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD/STORE	69
STACK OPS	0
ADD/SUBTRACT	15
MULTIPLY	1
BRANCH	34
TRANSFER	0
MISCELLANEOUS	5
	124

4.5.3 Quadratic (QUAD)

a. Procedure

T0) Select 8 sets of integers, (A_k, B_k, C_k, X_k), at random from the indicated intervals:

$$A_k,\ B_k,\ C_k,\quad 0 \leqq X \leqq 2^{15} - 1$$

$$X_k,\quad -10 \leqq X_k \leqq 10\ .$$

For each fault:

T1) Preset the program counter to the address of the first instruction.

T2) Store the (A_k, B_k, C_k, X_k) in successive locations of memory.

T3) Compute and store in successive locations of memory (overflows are ignored):

$$S_k = (A_k X_k) X_k - B_k X_k - C_k$$

$$K = 1, 2, \ldots., 8 .$$

T4) When the non-failed processor completes its last instruction, compare S_1, S_2, ..., S_8, term by term, in both the non-failed and failed processors. If S_k is the first variable to miscompare set $L = K$. If all S_k compare set $L = 0$.

b. Instruction Set

During a typical computation the following instructions were executed:

INSTRUCTION	FREQUENCY
LOAD/STORE	11
STACK OPS	4
ADD/SUBTRACT	3
MULTIPLY	3
BRANCH	6
TRANSFER	9
MISCELLANEOUS	5
	41

4.5.4 Flight Control System (FCS)

FCS was an existing 3-axis, high performance flight control computation for an advanced aircraft. The program consisted of seven modules:

1. Pitch axis control law.
2. Left horizontal tail cmd (TLCMD).
3. Right horizontal tail cmd (TRCMD).
4. Yaw axis control law and rudder cmd (RCMD).
5. Roll axis control law.
6. Left flaperon cmd (FLCMD).
7. Right flaperon cmd (FRCMD).

All integrators were initialized to zero for each run and each sensor input was selected at random for each pass through the program. The comparators were located at the actuator commands, i.e., at TLCMD, TRCMD, RCMD, FLCMD and FRCMD.

Because of the extreme length of the FCS program (e.g., 2,200 instructions, 13,729 microcycles) it was not possible to run the entire program for eight repetitions for each of 1,000 faults, as was intended, initially. Instead, a compromise was reached in which the program would be run for a single repetition for each of 1,000 faults and for 8 repetitions for each of 200 faults.

- Single Repetition Experiments

  In order to introduce latency during a single repetition the program was executed in parts, with each part designated as a "quasi-repetition". These quasi-repetitions were:

  <u>Quasi-Repetition #1</u>

  .. Pitch control law.
  .. Left horizontal tail cmd (TLCMD)

  <u>Quasi-Repetition #2</u>

  .. Retaining pitch control law computations, right horizontal tail cmd (TRCMD).

  <u>Quasi-Repetition #3</u>

  .. Yaw axis control law.
  .. Rudder cmd (RCMD)

  <u>Quasi-Repetition #4</u>

  .. Roll control law.
  .. Left flaperon cmd (FLCMD)

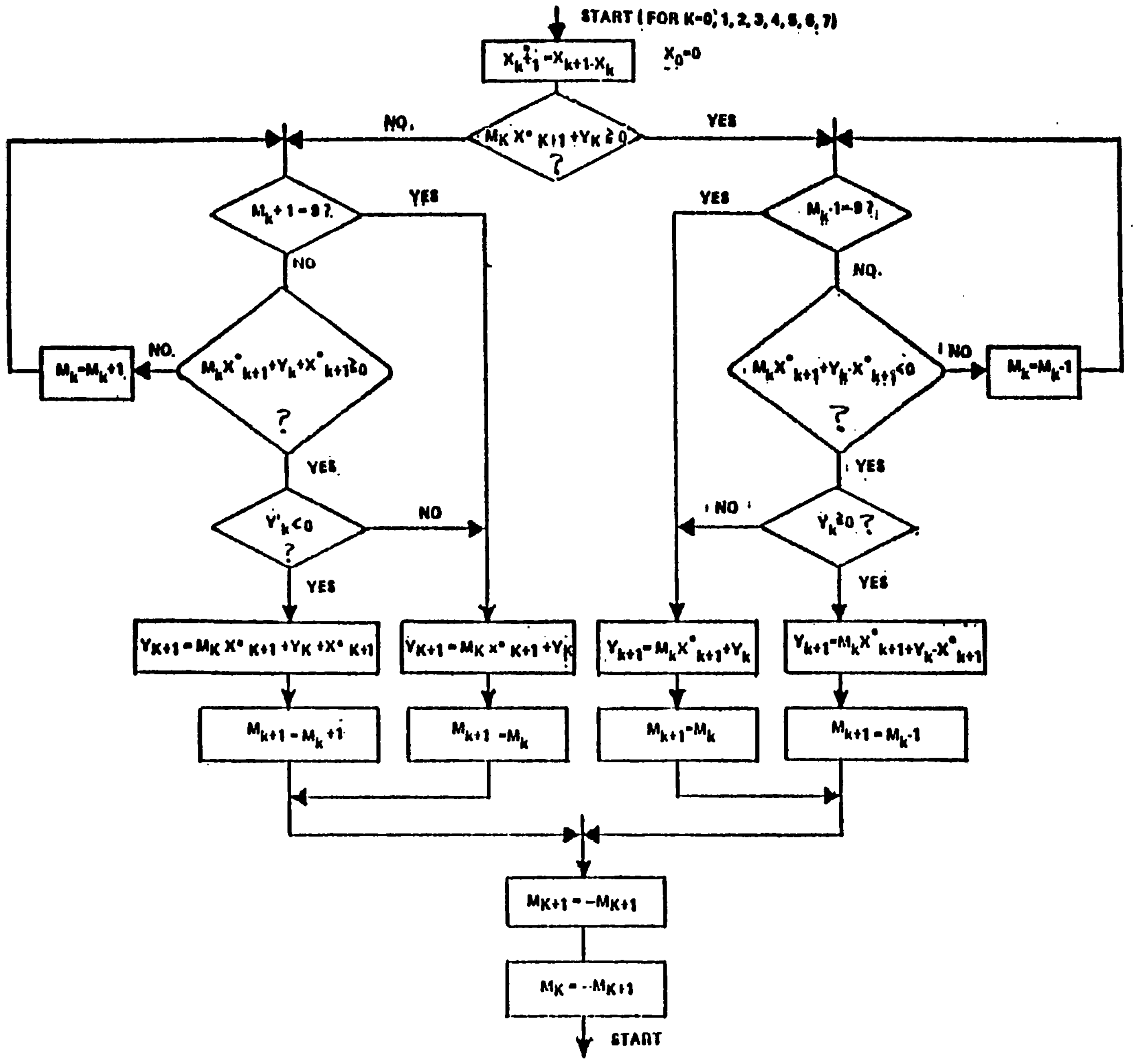
  <u>Quasi-Repetition #5</u>

  .. Retaining roll control law computations, right flaperon cmd (FRCMD).

- Eight Repetition Experiments

  In these experiments the seven modules (i.e., the five, quasi-repetitions) were executed for eight, complete repetitions for each injected fault.

In all of these experiments when the non-failed processor completes its last instruction the resultant commands are compared in both the non-failed and failed processors. Any discrepancy between corresponding commands designates a detected fault. When a fault is detected the preprocessor ignores detection in subsequent repetitions.



START (FOR K=0, 1, 2, 3, 4, 5, 6, 7)
$X^{\circ}_{k+1} = X_{k+1} - X_k$
$X_0 = 0$
$M_K X^{\circ}_{K+1} + Y_K \geq 0$?
NO
YES
$M_k + 1 = 8$?
YES
NO
$M_k X^{\circ}_{k+1} + Y_k + X^{\circ}_{k+1} \geq 0$?
NO
$M_k = M_k + 1$
YES
$Y'_k < 0$?
NO
YES
$Y_{K+1} = M_K X^{\circ}_{K+1} + Y_K + X^{\circ}_{K+1}$
$Y_{K+1} = M_K X^{\circ}_{K+1} + Y_K$
$M_{k+1} = M_k + 1$
$M_{k+1} = M_k$
$M_k - 1 = -8$?
YES
NO
$M_k X^{\circ}_{k+1} + Y_k - X^{\circ}_{k+1} < 0$?
NO
$M_k = M_k - 1$
YES
$Y_k \geq 0$?
NO
YES
$Y_{k+1} = M_k X^{\circ}_{k+1} + Y_k$
$Y_{k+1} = M_k X^{\circ}_{k+1} + Y_k - X^{\circ}_{k+1}$
$M_{k+1} = M_k$
$M_{k+1} = M_k - 1$
$M_{K+1} = -M_{K+1}$
$M_K = -M_{K+1}$
START

FIGURE 1 FLOW DIAGRAM FOR LINCON

5.0 RESULTS OF EXPERIMENTS

In this section the data from the experiments is presented concisely and with a minimum of commentary. A detailed analysis of the results is given in the next section.

5.1 Distribution of Faults

As indicated previously the selection of faults was random, with each device weighted in proportion to its failure rate. The failure rates of individual devices are given in Table 20. The failure rates of each partition of the CPU are given in Table 1.

The number of faults injected during each experiment are given in Table 2 for each of the four programs. The number of faults injected in each partition are given in Table 3. Once selected, the same faults were used in all experiments.

5.2 Experiments

To simplify the presentation material graphs and latency distributions will only be given for combined faults, irrespective of partition. However, the distributions for S-a-0, S-a-1 faults, by partitions, are given in tabular form.

For the purpose of comparison the latency distributions obtained from (ref. 1) are superimposed on the corresponding histograms obtained under the conditions of the present study. Also shown are the proportion of undetected faults after eight repetitions, corrected for indistinguishable faults. It is noted in Section 7.0 that, based on an analysis of 6600 faults, the proportion of indistinguishable gate faults is 16.5%.

5.2.1 SERCOM Experiment

After each injected fault SERCOM was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 2a, 2b. Tabular results are given in Table 4.

Figure 2a, Summarized
(Combined, gate-level faults)

- 53.6% undetected after a single repetition.
- 46.4% detected in the 1st repetition.
- 45% undetected after 8 repetitions.
- 34.1% undetected after 8 repetitions when corrected for indistinguishable faults.

Figure 2b, Summarized

(Combined, Component-level Faults)

- 31.8% undetected after a single repetition.
- 68.2% detected in the 1st repetition.
- 19.2% undetected after 8 repetitions.
- 3.2% undetected after 8 repetitions when corrected for indistinguishable faults.

5.2.2 LINCON Experiment

After each injected fault LINCON was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 3a, 3b. Tabular results are given in Table 5.

Figure 3a, Summarized

(Combined, gate-level faults)

- 46.5% undetected after a single repetition.
- 53.5% detected after a single repetition.
- 44.9% undetected after 8 repetitions.
- 34% undetected after 8 repetitions when corrected for indistinguishable faults.

Figure 3b, Summarized

(Combined, component-level faults)

- 19.2% undetected after a single repetition.
- 80.8% detected after a single repetition.
- 18.7% undetected after 8 repetitions
- 2.6% undetected after 8 repetitions when corrected for indistinguishable faults.

5.2.3 QUAD Experiment

After each injected fault QUAD was executed for 8 repetitions. The resultant histograms of detected faults versus repetitions to detection are shown in Figures 4a, 4b. Tabular results are given in Table 6.

Figure 4a, Summarized

(Combined, gate-level faults)

- 49.3% undetected after a single repetition.
- 50.7% detected after a single repetition.
- 41.3% undetected after 8 repetitions.
- 29.7% undetected after 8 repetitions when corrected for indistinguishable faults.

Figure 4b, Summarized

(Combined, component-level faults)

- 23.6% undetected after a single repetition.
- 76.4% detected after a single repetition.
- 17.1% undetected after 8 repetitions.
- 0.72% undetected after 8 repetitions when corrected for indistinguishable faults.

5.2.4 FCS Experiments (Quasi-Repetitions)

After each injected fault FCS was executed for a single repetition. However, as described in Section 4.5.4, the program was executed in 5 parts, designated as quasi-repetitions. The resultant histograms are shown in Figures 5a, 5b. Tabular results are given in Table 7.

Figure 5a, Summarized

(Combined, gate-level faults)

- 43% undetected after quasi-repetition #1.
- 57% detected after quasi-repetition #1.
- 41.9% undetected after a complete pass.

- 30.4% undetected after a complete pass when corrected for indistinguishable faults.

Figure 5b, Summarized

(Combined, component-level faults)

- 16% undetected after quasi-repetition #1.
- 84% detected after quasi-repetition #1.
- 15.8% undetected after a complete pass.
- 0% undetected after a complete pass when corrected for indistinguishable faults.

It is noted that, once a fault has been detected, the preprocessor ignores detection in subsequent repetitions. This is the reason, for example, that Quasi-Repetitions #2, #3, #4 and #5 show poor coverage relative to Quasi-Repetition #1 in Figure 5, even though the computations are similar.

5.2.5 FCS Experiments (True-Repetitions)

After each injected fault FCS was executed for 8 repetitions. The resultant histogram is shown in Figure 6. Tabular results are given in Table 8.

Figure 6, Summarized

(Combined, gate-level faults)

- 37.5% undetected after a single repetition.
- 62.5% detected after a single repetition.
- 34% undetected after 8 repetitions.
- 21% undetected after 8 repetitions when corrected for indistinguishable faults.

5.3 Urn Model Parameters

The parameters of the Urn Model were estimated for SERCOM, LINCON, QUAD and FCS using the estimators defined in Section 9.3 for combined, S-A-0 and S-A-1 faults. The resultant estimates of a, P, P_0, as defined in Sections 8.0 and 9.3, are given in Table 9. All estimates were obtained using 8 repetitions of each program.

As an illustration of the Urn Model "fit" Figure 7 shows the resultant Urn Model distribution superimposed on the empirical distribution for FCS.

5.4 Accuracy and Confidence of Results

The accuracy of coverage estimates will be given for combined, gate-level faults, only. The estimates are based on the total set of faults irrespective of their distinguishability.

SERCOM (1000 Faults)

After 8 repetitions 55% of all faults were detected. The error, at the 95% confidence level, is

$$\varepsilon = 1.96 \sqrt{\frac{.55(.45)}{1000}} = .0308 \ (3.08\%)$$

LINCON (1000 Faults)

After 8 repetitions 55.1% of all faults were detected. The error, at the 95% confidence level is

$$\varepsilon = .0308 \ (3.08\%)$$

QUAD (1000 Faults)

After 8 repetitions 58.7% of all faults were detected. The error, at the 95% confidence level, is

$$\varepsilon = 1.96 \sqrt{\frac{.587(.413)}{1000}} = .0305 \ (3.05\%)$$

FCS (200 Faults)

After 8 repetitions 66% of all faults were detected. The error, at the 95% confidence level, is

$$\varepsilon = 1.96 \sqrt{\frac{.66(.34)}{200}} = .066 \ (6.6\%)$$

Urn Model Parameters

The accuracy is illustrated for the QUAD Program.
There

$P \approx .864$
$Po \approx .587$
$a \approx .667$

From Section 9.4.3 the errors at the 95% confidence levels are

$$\epsilon_p = 1.96 \sqrt{\frac{.864(.136)}{1000 \times .587}} = .028 \ (2.8\%)$$

$$\epsilon_{p_o} = 1.96 \sqrt{\frac{.587(.413)}{1000}} = .031 \ (3.1\%)$$

$$\epsilon_a = 1.96 \sqrt{\frac{(.667)^2 \ (.333)}{1000 \times .587 \times .136}} = .084 \ (8.4\%)$$

TABLE 1

FAILURE RATES BY PARTITION

PARTITION		FAILURE RATE (MIL-HDBK-2175)	PROPORTION OF TOTAL
1		7.1014×10^{-6} /HR	.186
2		5.8223×10^{-6} /HR	.153
3		7.4706×10^{-6} /HR	.196
4		9.4863×10^{-6} /HR	.249
5		7.056×10^{-6} /HR	.185
6		1.1867×10^{-6} /HR	.031
	TOTALS	38.1233×10^{-6} /HR	1.0

TABLE 2

NUMBER OF FAULTS INJECTED

EXPERIMENT	GATE-LEVEL	COMPONENT LEVEL
SERCOM	1000	1000
LINCON	1000	1000
QUAD	1000	1000
FCS #1	1000	500
FCS #2	200	X

TABLE 3A

NUMBER OF GATE-LEVEL FAULTS INJECTED BY PARTITIONS

PROGRAMS: SERCOM, LINCON, QUAD

PARTITION	S-A-0	S-A-1	COMBINED
1	90	92	182
2	89	83	172
3	117	105	222
4	111	120	231
5	79	79	158
6	18	17	35
	504	496	1000

PROGRAM: FCS (1000 FAULT CASE)

PARTITION	S-A-0	S-A-1	COMBINED
1	80	89	169
2	77	76	153
3	106	106	212
4	126	101	227
5	102	103	205
6	18	16	34
	509	491	1000

TABLE 3B

NUMBER OF COMPONENT-LEVEL FAULTS INJECTED BY PARTITIONS

PROGRAMS: SERCOM, LINCON, QUAD

PARTITION	S-A-0	S-A-1	COMBINED
1	122	124	246
2	104	104	208
3	147	138	285
4	141	120	261
	514	786	1000

PROGRAM: FCS (500 FAULT CASE)

PARTITION	S-A-0	S-A-1	COMBINED
1	55	48	103
2	62	52	114
3	67	73	140
4	74	69	143
	258	242	500

TABLE 4a SERCOM LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	60	68	2	1	0	2	0	0	0	0	1	0	0	0	0	0	90	92
P_2	48	62	10	3	0	0	0	0	0	0	0	0	0	0	0	0	89	83
P_3	67	61	6	12	0	0	0	0	5	1	0	0	0	0	0	0	117	105
P_4	31	62	14	6	0	0	0	0	6	8	2	0	0	0	1	1	111	120
P_5	2	2	1	1	0	0	0	0	1	0	0	0	0	0	0	0	79	79
P_6	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	17	18
TOTAL	209	255	34	24	0	2	0	0	12	9	3	0	0	0	1	1	503	497

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 4b   SERCOM LATENCY DATA

COMPONENT LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	94	97	3	8	0	3	0	0	0	0	0	0	0	0	0	0	120	126
P_2	60	79	14	6	0	0	0	0	0	0	0	0	0	0	0	0	104	104
P_3	110	104	14	4	0	0	2	3	0	0	0	0	0	0	0	2	159	153
P_4	69	69	25	14	16	9	0	0	0	0	0	0	0	0	3	0	129	105
P_5																		
P_6																		
TOTAL	333	349	56	32	16	12	2	3	0	0	0	0	0	0	3	2	512	488

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 5a LINCON LATENCY DATA

GATE-LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	68	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	92
P_2	66	65	0	0	0	0	0	0	0	0	0	0	0	0	0	0	89	83
P_3	66	69	0	0	0	0	0	0	0	0	0	0	0	1	0	0	117	105
P_4	57	68	1	2	1	0	1	0	0	2	0	1	1	2	1	1	111	120
P_5	2	3	1	0	0	0	0	0	0	0	0	0	0	1	0	0	79	79
P_6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	18
TOTAL	260	275	2	2	1	0	1	0	0	2	0	1	1	4	1	1	503	497

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 5b LINCON LATENCY DATA

COMPONENT LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	105	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	120	126
P_2	83	85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	104	104
P_3	96	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	147	138
P_4	118	106	3	0	0	0	0	0	0	0	1	1	0	0	0	0	142	119
P_5																		
P_6																		
TOTAL	402	406	3	0	0	0	0	0	0	0	1	1	0	0	0	0	513	487

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 6a QUAD LATENCY DATA

GATE LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	58	63	3	1	0	3	1	1	0	0	0	0	0	0	2	1	90	92
P_2	58	61	10	4	2	1	0	0	0	0	0	0	0	0	0	0	89	83
P_3	70	67	8	5	0	0	0	0	0	0	0	0	0	0	0	0	117	105
P_4	54	70	14	9	5	3	0	2	0	0	0	0	0	0	0	0	111	120
P_5	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	79	79
P_6	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	18	17
TOTAL	243	264	38	21	7	7	1	3	0	0	0	0	0	0	2	1	504	496

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 6b QUAD LATENCY DATA

COMPONENT LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	92	93	3	8	0	3	0	0	0	0	0	0	0	0	4	0	122	124
P_2	68	82	13	4	3	1	0	0	0	0	0	0	0	0	0	0	104	104
P_3	98	100	13	1	0	0	0	0	0	0	0	0	0	0	0	0	147	138
P_4	126	105	8	2	0	1	0	1	0	0	0	0	0	0	0	0	141	120
P_5																		
P_6																		
TOTAL	384	380	37	15	3	5	0	1	0	0	0	0	0	0	4	0	514	486

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 7a FLIGHT CONTROL COMPUTATIONS LATENCY DATA
(QUASI-REPETITIONS)

GATE LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	67	77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80	89
P_2	61	74	0	0	0	0	0	0	0	0	0	0	0	0	0	0	77	76
P_3	71	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	106	106
P_4	61	64	3	0	6	1	0	0	0	0	0	0	0	0	0	0	126	101
P_5	11	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	102	103
P_6	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	18	16
TOTAL	272	298	4	0	6	1	0	0	0	0	0	0	0	0	0	0	509	491

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 7b FLIGHT CONTROL COMPUTATIONS LATENCY DATA
(QUASI-REPETITIONS)

COMPONENT LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	51	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	55	48
P_2	54	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	62	52
P_3	44	55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67	73
P_4	65	61	0	0	1	0	0	0	0	0	0	0	0	0	0	0	74	69
P_5																		
P_6																		
TOTAL	214	206	0	0	1	0	0	0	0	0	0	0	0	0	0	0	258	242

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 8 FLIGHT CONTROL COMPUTATION LATENCY DATA (TRUE REPETITIONS)

GATE LEVEL FAULTS

PARTITION	DETECTED FAULTS																FAULTS INJECTED	
	M_1	N_1	M_2	N_2	M_3	N_3	M_4	N_4	M_5	N_5	M_6	N_6	M_7	N_7	M_8	N_8	M	N
P_1	15	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	15
P_2	9	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	17
P_3	14	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	21
P_4	18	22	5	1	0	0	0	0	0	0	0	0	0	0	0	0	29	27
P_5	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	17	17
P_6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3
TOTAL	58	67	5	1	1	0	0	0	0	0	0	0	0	0	0	0	100	100

M_i = Detected S-a-0 Faults, ith Cell

N_i = Detected S-a-1 Faults, ith Cell

TABLE 9

URN MODEL PARAMETER ESTIMATES FOR GATE-LEVEL FAULTS

		a	P	Po
SERCOM	COMBINED	.4914	.8436	.550
	S-A-0	.325	.807	.515
	S-A-1	.507	.876	.586
LINCON	COMBINED	.2424	.971	.551
	S-A-0	.3	.9774	.5288
	S-A-1	.2173	.9649	.5734
QUAD	COMBINED	.667	.8637	.587
	S-A-0	.6956	.835	.577
	S-A-1	.6274	.892	.597
FCS	COMBINED	.875	.947	.66
	S-A-0	.857	.906	.64
	S-A-1	1.0	.985	.68

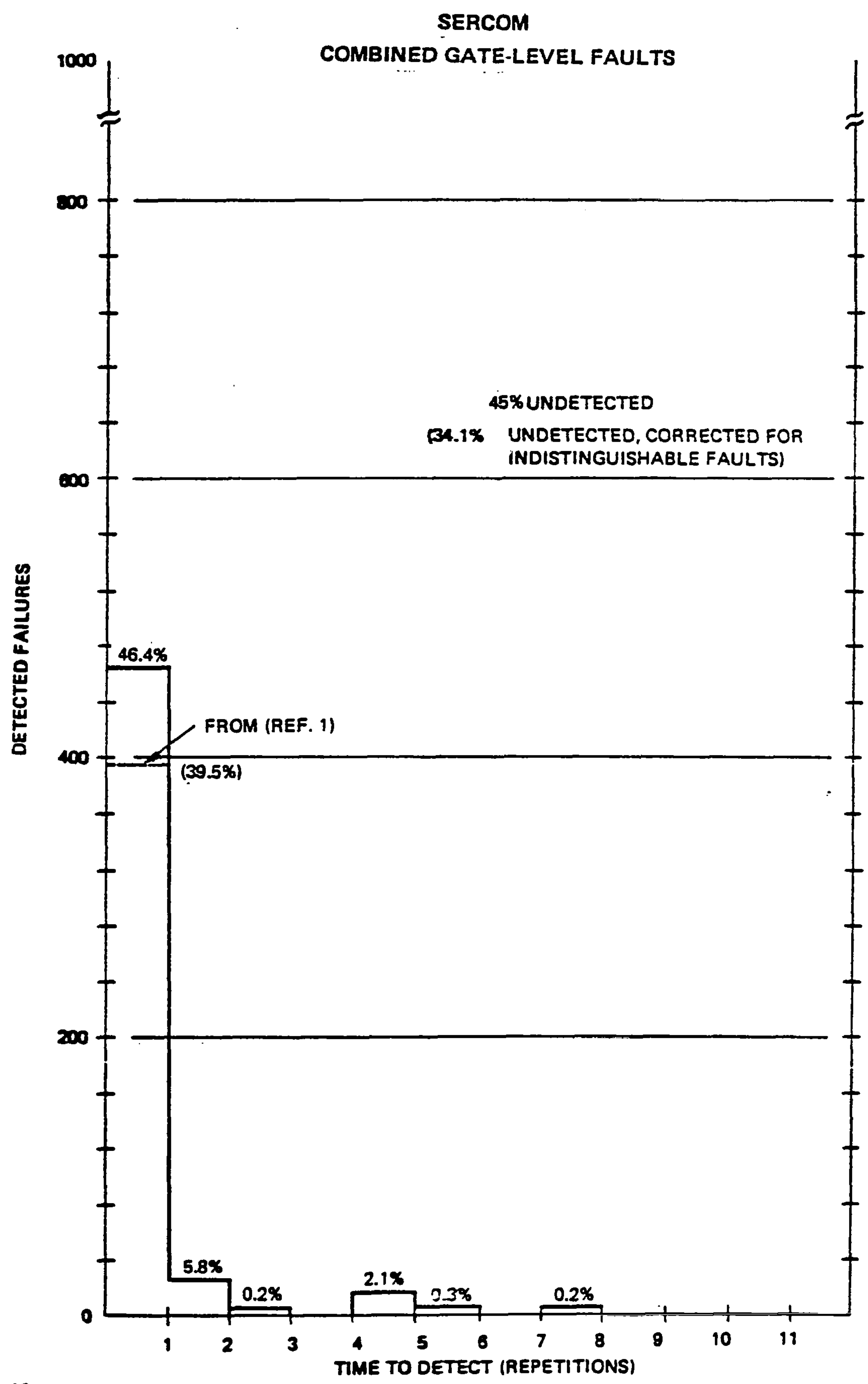

SERCOM
COMBINED GATE-LEVEL FAULTS
1000
800
600
400
200
0
DETECTED FAILURES
45% UNDETECTED
(34.1% UNDETECTED, CORRECTED FOR INDISTINGUISHABLE FAULTS)
46.4%
FROM (REF. 1)
(39.5%)
5.8%
0.2%
2.1%
0.3%
0.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 2A

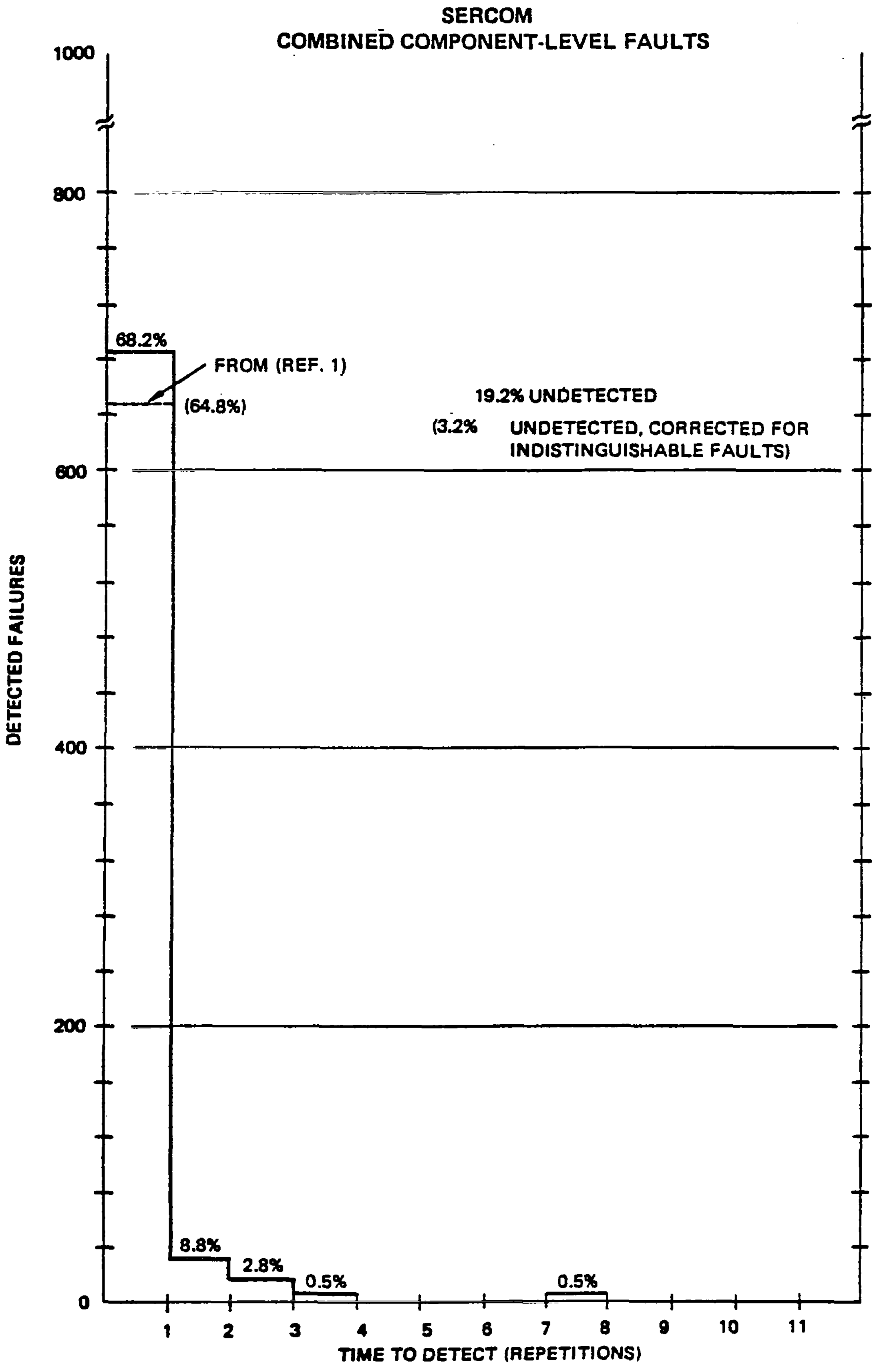

SERCOM
COMBINED COMPONENT-LEVEL FAULTS
1000
800
600
400
200
0
68.2%
FROM (REF. 1)
(64.8%)
19.2% UNDETECTED
(3.2% UNDETECTED, CORRECTED FOR INDISTINGUISHABLE FAULTS)
8.8%
2.8%
0.5%
0.5%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)
DETECTED FAILURES

FIGURE 2B

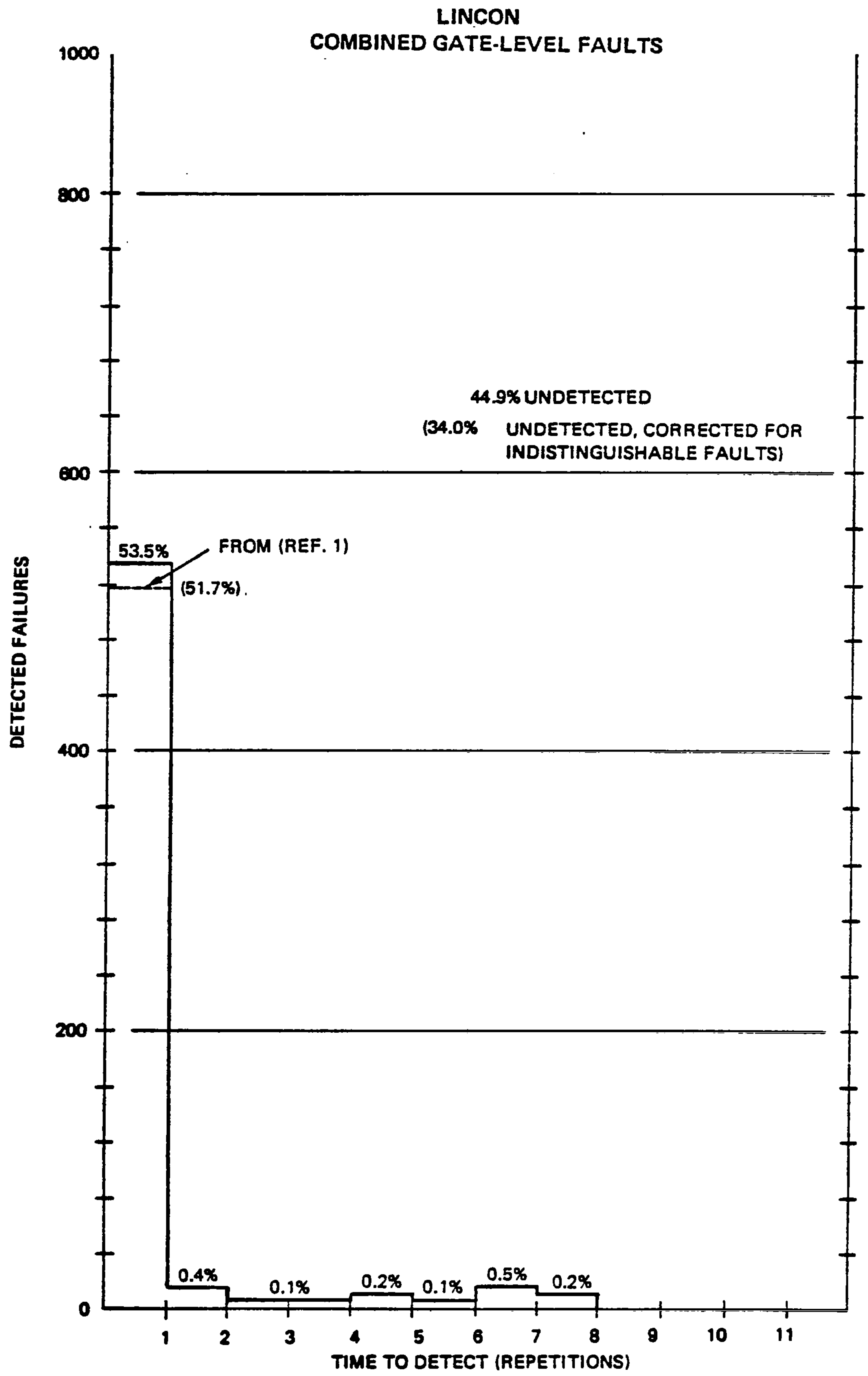

LINCON
COMBINED GATE-LEVEL FAULTS
1000
800
600
400
200
0
DETECTED FAILURES
44.9% UNDETECTED
(34.0% UNDETECTED, CORRECTED FOR INDISTINGUISHABLE FAULTS)
53.5%
FROM (REF. 1)
(51.7%).
0.4%
0.1%
0.2%
0.1%
0.5%
0.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 3A

LINCON
COMBINED COMPONENT-LEVEL FAULTS

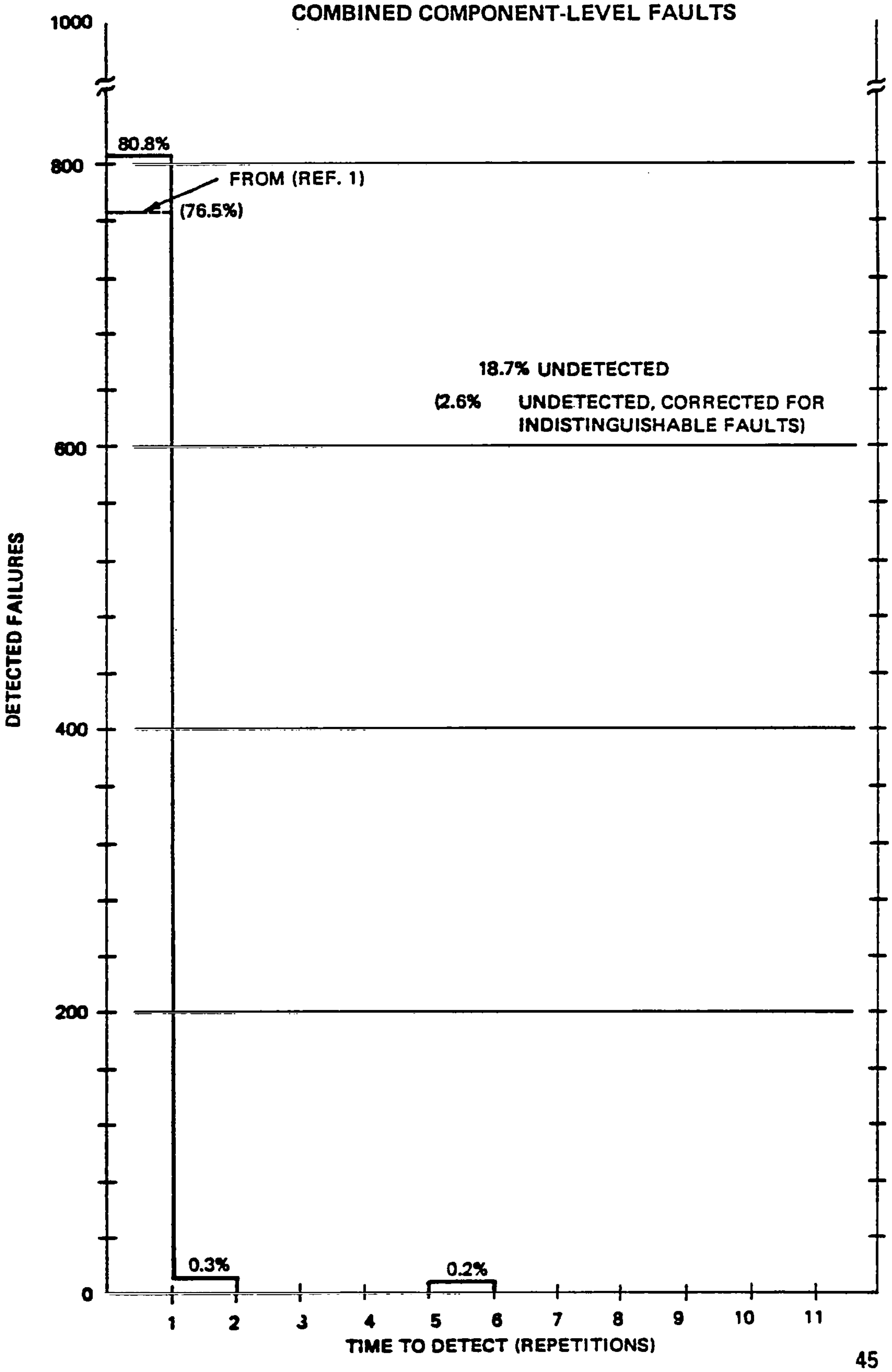


FIGURE 3B

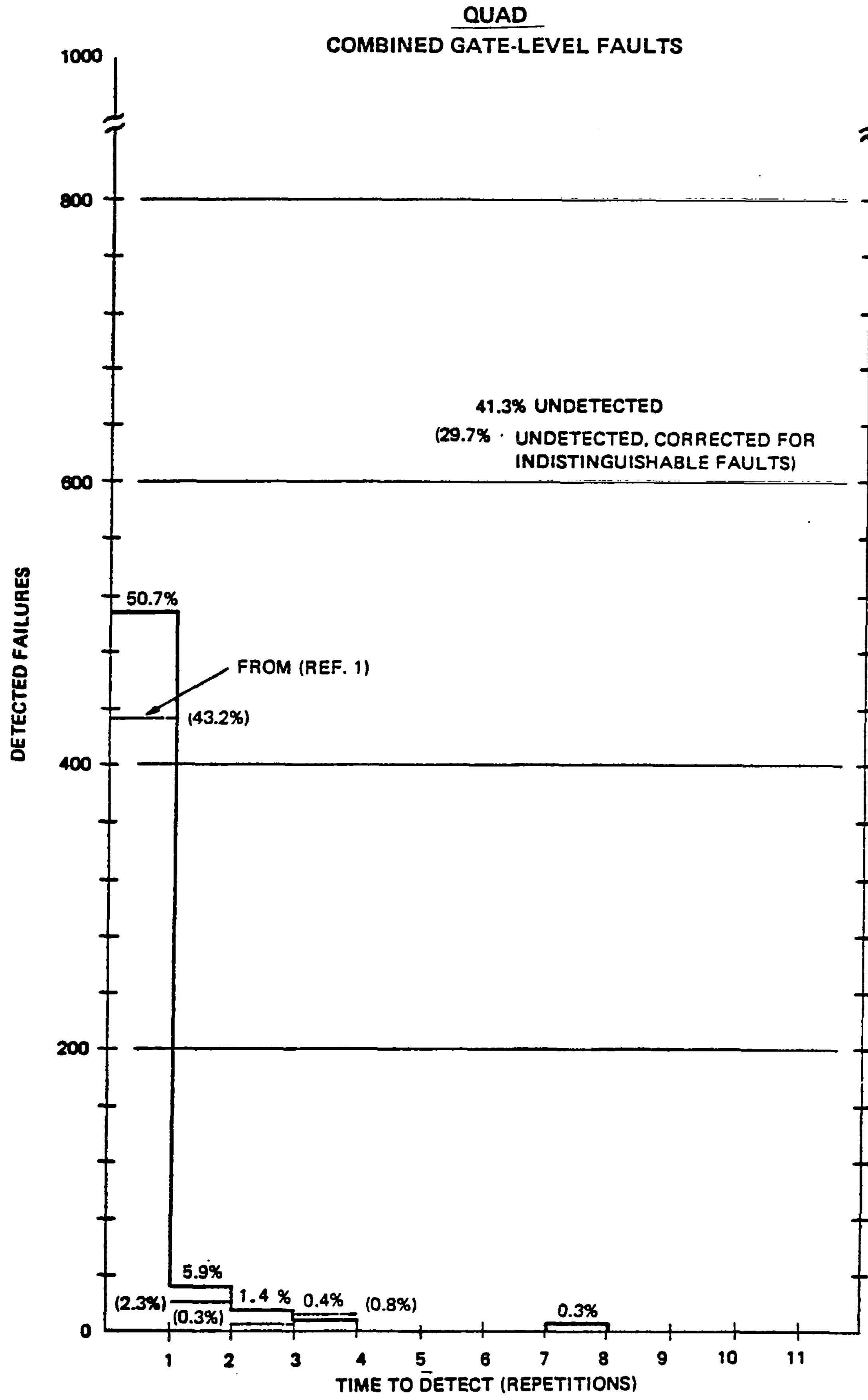

QUAD
COMBINED GATE-LEVEL FAULTS
1000
800
600
400
200
0
DETECTED FAILURES
41.3% UNDETECTED
(29.7% · UNDETECTED, CORRECTED FOR INDISTINGUISHABLE FAULTS)
50.7%
FROM (REF. 1)
(43.2%)
5.9%
(2.3%)
1.4 %
(0.3%)
0.4%
(0.8%)
0.3%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (REPETITIONS)

FIGURE 4A

OUAD
COMBINED COMPONENT-LEVEL FAULTS

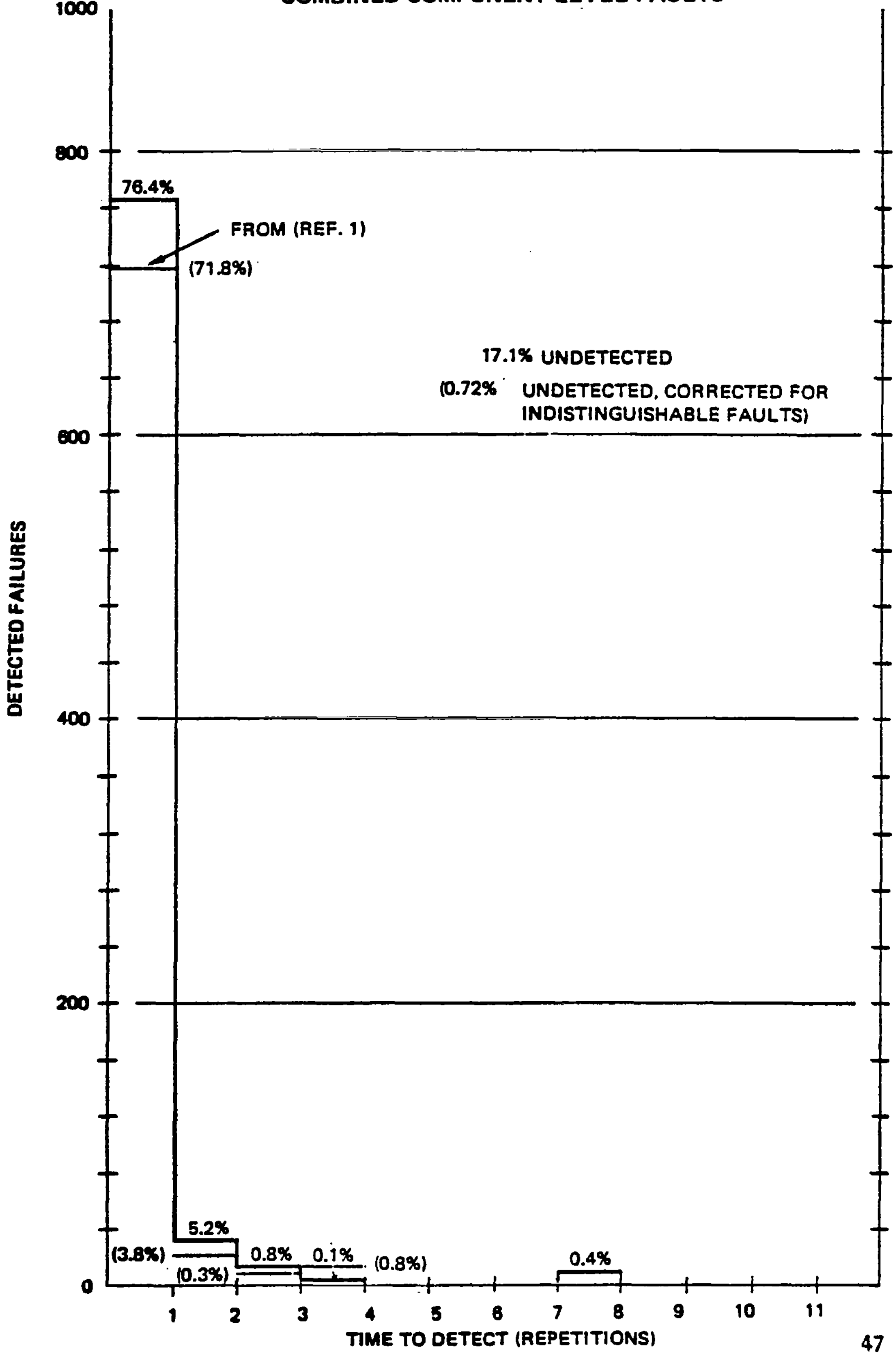


FIGURE 4B

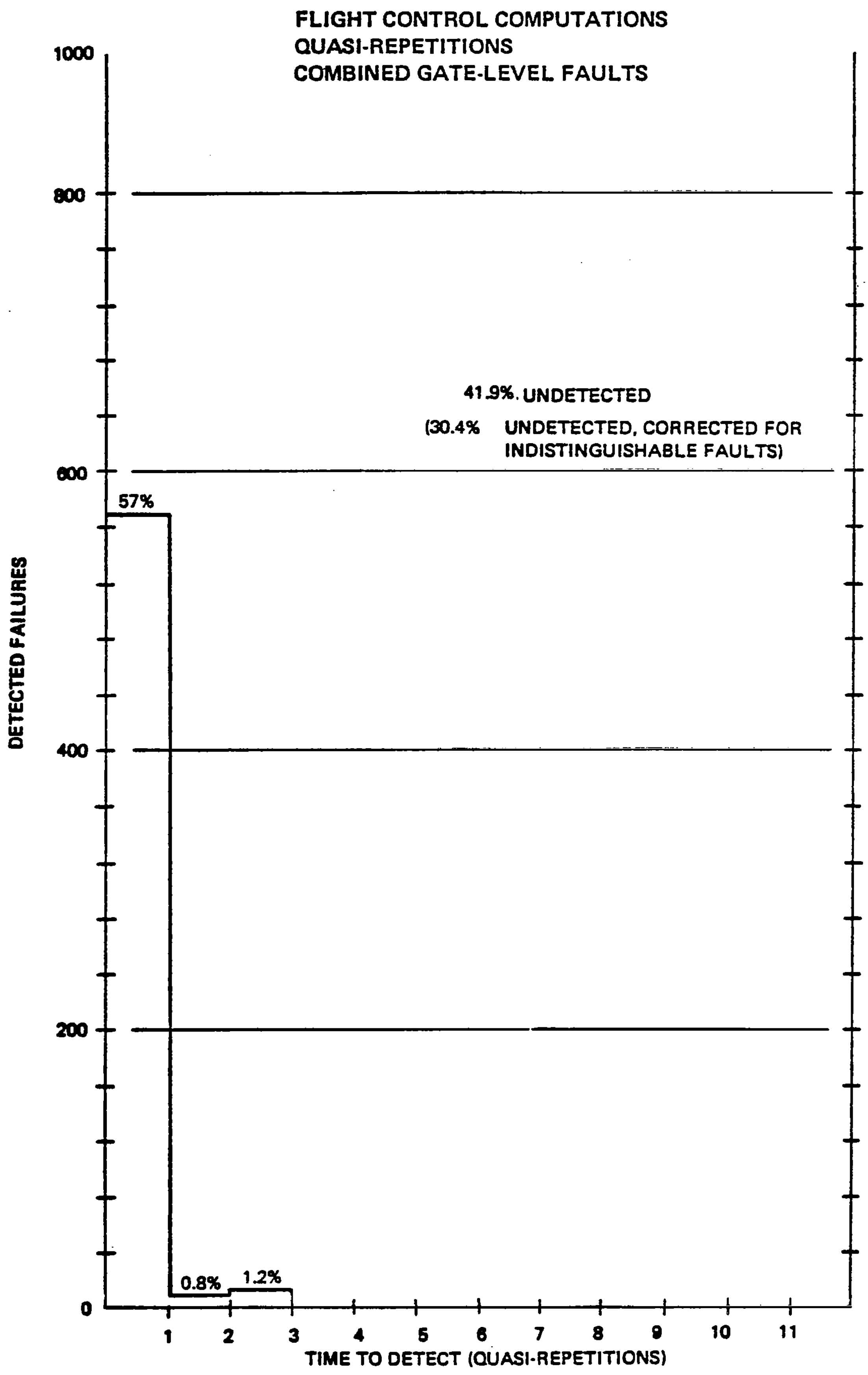

FLIGHT CONTROL COMPUTATIONS
QUASI-REPETITIONS
COMBINED GATE-LEVEL FAULTS
1000
800
600
400
200
0
DETECTED FAILURES
41.9%. UNDETECTED
(30.4% UNDETECTED, CORRECTED FOR INDISTINGUISHABLE FAULTS)
57%
0.8%
1.2%
1
2
3
4
5
6
7
8
9
10
11
TIME TO DETECT (QUASI-REPETITIONS)

FIGURE 5A

FLIGHT CONTROL COMPUTATIONS
QUASI-REPETITIONS
COMBINED COMPONENT-LEVEL FAULTS

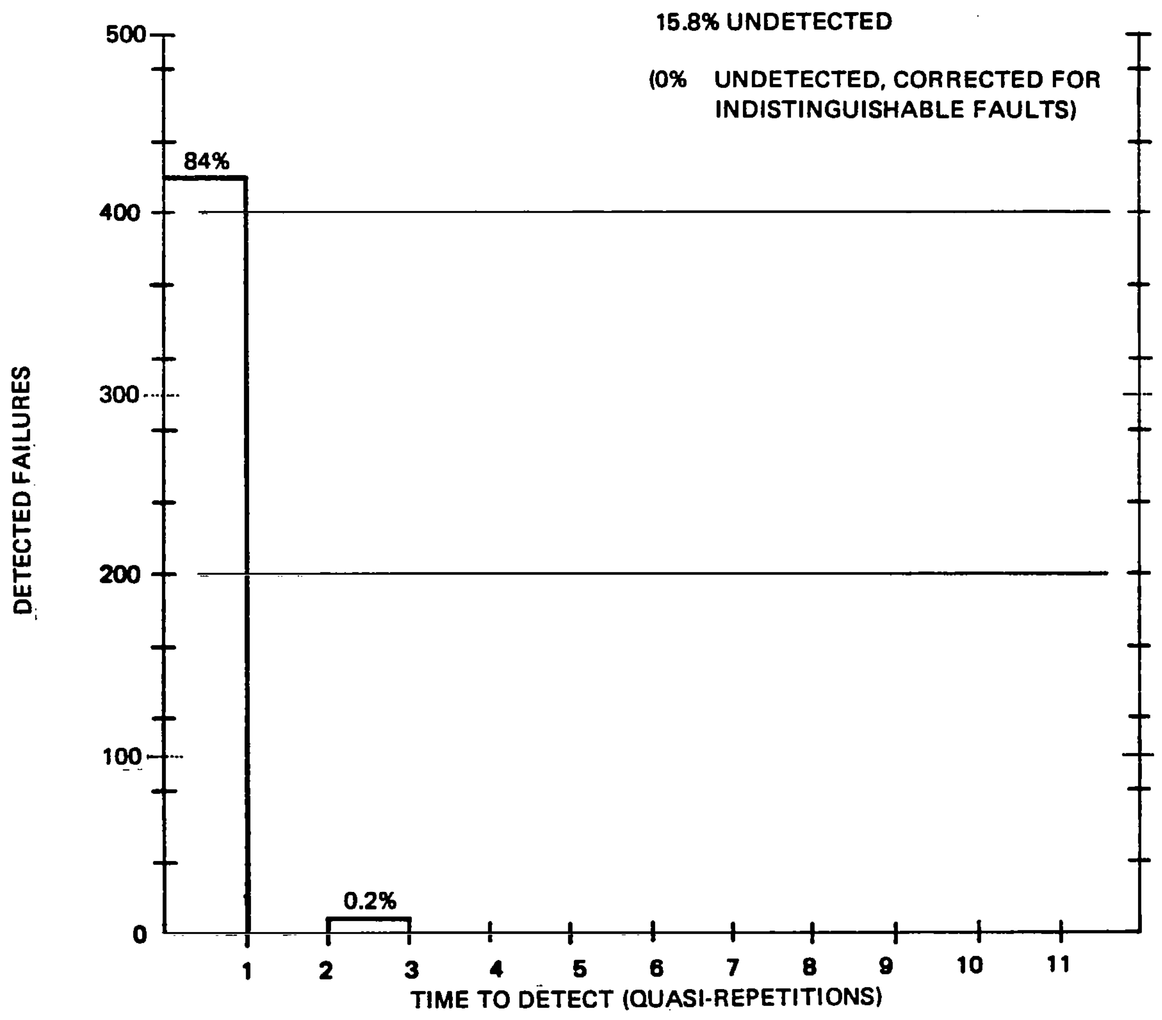


FIGURE 5B

FLIGHT CONTROL COMPUTATIONS
TRUE REPETITIONS
COMBINED GATE-LEVEL FAULTS

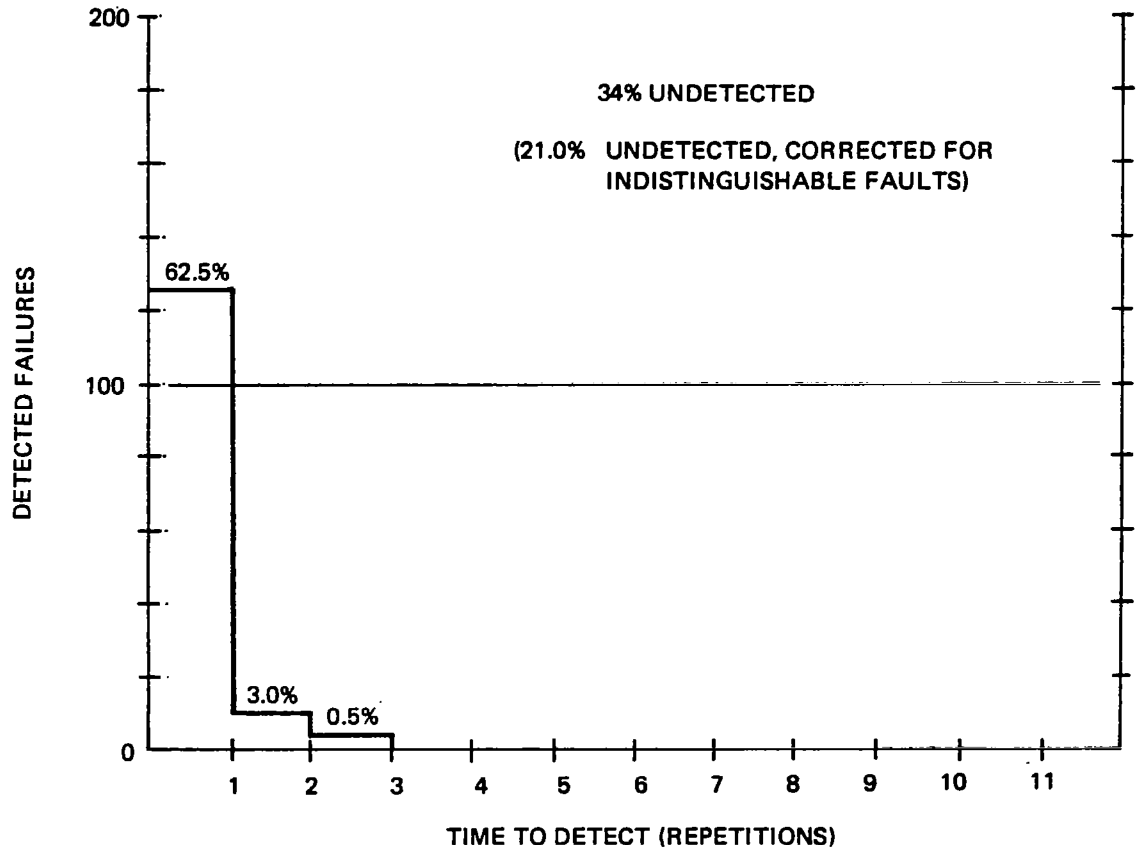


FIGURE 6

URN MODEL DISTRIBUTION
FLIGHT CONTROL COMPUTATIONS
TRUE REPETITIONS
COMBINED GATE-LEVEL FAULTS

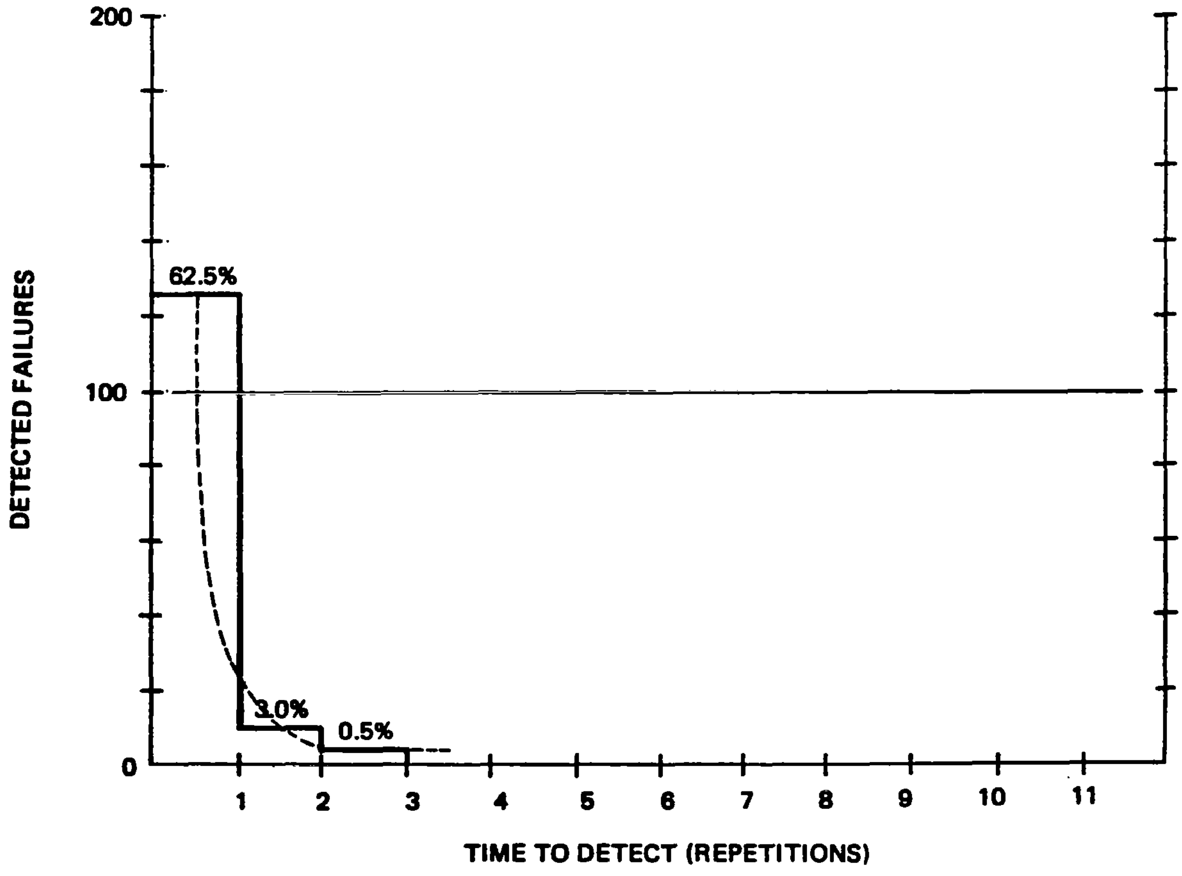


FIGURE 7

6.0 SUMMARY OF EXPERIMENTS

6.1 Latency

- Most detected faults are detected in the first repetition. Subsequent repetitions do not appreciably increase the proportion of detected faults. However, it appears that short programs have a tendency to benefit more from subsequent repetitions than lengthier programs (i.e., FCS). It is conjectured that shorter programs rely more heavily on inputs for failure mode excitation than their lengthier counterparts.

- S-a-1 faults are easier to detect than S-a-0 faults.

- The micromemory (i.e., Partition #5) contains a large proportion of undetected faults.

- A large proportion of faults remain undetected after as many as 8 repetitions. For example, in a flight control computation of 2200 instructions 21% of distinguishable gate-level faults remained undetected.

- Component-level faults are easier to detect than gate-level faults. As an example, Table 10 summarizes detection coverage of distinguishable gate-level faults for 8 repetitions of SERCOM, LINCON, QUAD and FCS. Also shown are the respective coverages of the final self-test program described in Section 7.0.

- The latency distributions for SERCOM, LINCON, QUAD and FCS are uncorrected for indistinguishable faults. A detailed analysis of 6600 faults (see Section 7.0) indicates that the proportion of indistinguishable (i.e., "don't care") faults in the BDX-930 is 16.5%. The appropriate correction factors can be obtained by the method described in Section 4.3. Table 10 summarizes detection coverage of distinguishable faults.

- Based on our experience with self-test (see Section 7.0) it may be concluded that a program's ability to detect faults cannot be characterized by the number of instructions or the instruction mix. As a consequence, it is not surprising that a short program such as QUAD (41 instructions) has a coverage of 70.3%, while a lengthy program such as FCS (2200 instructions) has a coverage of only 79.0%.

- The distributions for LINCON are unique in that a very small proportion of faults are detected in the second and subsequent repetitions. In this respect it is similar to the distributions for the flight control system when the latter was subdivided into quasi-repetitions (see Figures 5a, 5b). In these experiments each program was executed, effectively, for a single repetition. The other programs, on the other hand, were executed repeatedly and in their entirety with a different set of inputs for each pass. As a consequence, we believe that the distribution for LINCON and the "QUASI FCS" are not representative.

- Based upon these results for LINCON and QUASI FCS it appears that the excitation supplied by inputs accounts for most of the coverage in the second and subsequent repetitions.

- Detection coverage between the second and last repetitions varied between 3.5% (FCS) and 8.6% (SERCOM). For the programs of the previous study these coverages were:

FETSTO:	8.4%
FIB:	7.0%
ADDSUB:	7.0%

- It is interesting to compare the gate-level latency distributions for FETSTO, FIB and ADDSUB, obtained from the previous study, with those for SERCOM, QUAD and FCS of the present study. The number of executable instructions in each program are:

FETSTO:	6
FIB:	11
ADDSUB:	11
SERCOM:	44
QUAD:	41
FCS:	2200

The first three programs were limited to a simple instruction set, whereas the last three used a variety of "high powered" instructions.

The respective latency distributions are shown in Figure 8. From the figure it can be seen that the distributions are qualitatively similar despite the dissimilarity of their programs. The major difference is the distribution between coverage of the first repetition and total coverage. It appears that lengthier programs yield a greater coverage in the first repetition than shorter programs.

6.2 Urn Model

- The Urn Model can, at least qualitatively, characterize the shape of a latency distribution. This can be attributed to (1) the monotonic decreasing property of the empirical distribution and (2) the three degrees-of-freedom which the Urn Model provides for a best fit.

- It is doubtful that the Urn Model parameters can be predicted for a program on the basis of length or instruction mix.

- Table 11 summarizes the Urn Model parameters for combined, gate-level faults for all of the programs of this and the previous study. Based on these results, we make the following observations:

  .. The Urn Model parameters are in general agreement with the empirical distributions.

  .. In every case, P_0, the probability that a fault is detected, eventually, coincided with coverage after 8 repetitions.

  .. In every case, P_1, the probability that a fault is detected in the first repetition, coincided with the empirical distribution.

  .. In every case, $P_0(1-P)$, the probability that a fault is detected in subsequent repetitions, was in close agreement with the empirical distribution.

- Based upon the results from the LINCON and QUASI FCS experiments we conjecture that, if the inputs are invariant, almost all detected faults are detected in the first repetition and coverage during subsequent repetitions will be negligible. As a consequence, the observed actual coverage during subsequent repetitions must be due to varying inputs.

  The parameter, a, which gives the Urn Model distribution its exponential character, varies widely. It appears that "a" is a function primarily of input excitation and is a measure of the effectiveness of this excitation in fault detection.

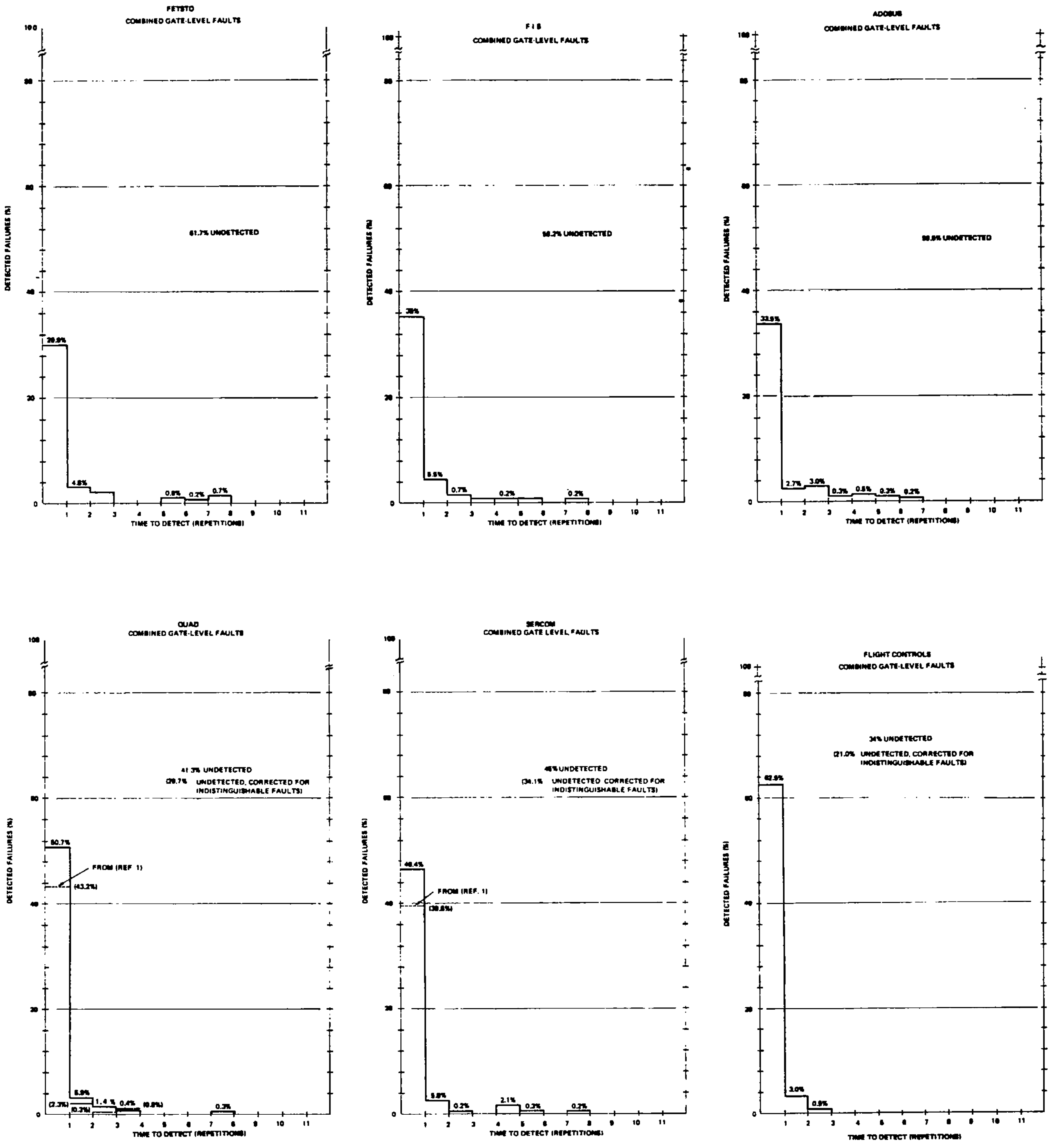


FIGURE 8. COMPARISON OF FAULT LATENCY DISTRIBUTIONS

TABLE 10

COMPARISON OF GATE VERSUS COMPONENT-LEVEL COVERAGE *

Program	Gate-Level Coverage**	Component-Level Coverage**
SERCOM	.659	.968
LINCON	.66	.974
QUAD	.703	.9928
FCS	.79	1.0
Final Self-Test	.92	.976

* Coverages have been corrected for indistinguishable faults

** After 8 repetitions

TABLE 11

URN MODEL PARAMETERS
COMBINED, GATE-LEVEL FAULTS

Program	# Words	a	P	Po	PoP*	Po(1-P)**
FETSTO	6	.4386	.7776	.3845	.3	.086
FIB	11	.6823	.8366	.4184	.35	.0684
ADDSUB	11	.4691	.8254	.4058	.33	.071
SERCOM	44	.4914	.8436	.55	.46	.086
LINCON	124	.2424	.971	.551	.535	.016
QUAD	41	.667	.8637	.587	.51	.08
FCS	2200	.875	.947	.66	.625	.04

* PoP = P_1 = proportion of faults detected in the first repetition

** $P_0(1-P)$ = proportion of faults detected in subsequent repetitions

7.0 SELF-TEST DESIGN AND VALIDATION

7.1 Initial Self-Test Program

The initial self-test program was based on the belief that coverage could be achieved by exercising a sufficiently large set of instructions irrespective of their mechanization in hardware. As subsequent events proved, however, this approach resulted in an inefficient self-test in that it tended to exercise some hardware repeatedly while omitting to exercise a substantial proportion of the remainder. Undoubtedly, the addition of more instructions would eventually have achieved the desired coverage but at the cost of run-time and memory. As a consequence, it was decided to redesign the self-test program.

7.2 Principal Tests

Based on an analysis of undetected faults in the initial self-test the major effort was directed at the following hardware elements:

- Scratch Pad of 2901 (i.e., 16 accumulators).
- 9407 Address Processor.
- Arithmetic Logic Unit of 2901.
- Micromemory.

The gate-level equivalent circuits of these elements were analyzed to determine which instructions and instruction sequences were most effective in exercising the component gates. Since the test sequences were hardware intensive and directed specifically at the BDX-930 computer a description of each test would not be very illuminating to the reader, and would, moreover, require a detailed analysis of the data paths exercised by each BDX-930 instruction. However, because the micromemory is generic to all computers and presented the greatest challenge to fault detection, a brief description of the micromemory test will be given.

Micromemory Test

The microprogram memory consists of 512, 56-bit microinstructions. However, only 382 microinstructions are used. Moreover, the last three bits of each word are also unused. Thus, the proportion of indistinguishable faults is at least 29.4%.

All of the microinstructions used in previous tests were analyzed to determine which microinstruction remained unexercised. Additional instructions were added to exercise micromemory instruction coverage. The final microinstruction set exercised 45.25% of the microinstructions. However, this did not mean that an equivalent proportion of faults would be detected. The problem here was that many microinstructions contain branching conditions which could only be exercised by multiple calls of the microinstruction.

In connection with micromemory fault detection two problems were paramount:

1. High coverage required the execution of a large number of micro-instructions with an attendant increase in memory and real time.

2. It was extremely difficult to identify indistinguishable faults. With no failure present it was relatively easy to identify micromemory bits that, by design, had no effect on operations. However, this was not the case when a nominally "don't care" bit was in a failed state. The identification process required the expertise of a computer designer, and even so, it was a time comsuming process.

7.3 Self-Test Results

The design of the self-test program was an evolutionary process. After each update the resultant coverage was estimated via the emulator. Undetected faults were analysed and the test again updated.

The initial self-test consisted of 1,100 words and required 7,777 micro-cycle to complete. The test was reviewed to determine the most effective tests. These were retained and the rest discarded. The first, modified test consisted of 192 words and required 974 microcycles. After each revision a new fault set was selected and faults were injected at the gate-level. All successive revisions merely added new tests to those of its predecessors. As a consequence, coverage could only improve with each revision.

The successive revisions and their corresponding coverages are tabulated in Table 12. The development process was terminated following Revision 8. The non-monotonic coverage is the result of statistical fluctuation. Table 13 shows the gate-level coverage of Revision 8 by partitions. It is interesting to compare these results with those of the initial self-test, which are given in Table 14. The accuracy for 3,000 faults is $\pm 1\%$ with 95% confidence. Table 15 shows the component-level coverage of Revision 8 by partitions. The corresponding results of the initial self-test are given in Table 16.

Examination of coverage by partitions (Table 13) indicates that the worst coverage was in the micromemory and control proms. It is clear that the only way to improve coverage of these devices is to increase the number of microinstructions executed by self-test. Since the conservation of memory was an important design goal this approach was rejected.

Another reason for poor coverage of the micromemory was the way that the unit was emulated, i.e., at the cell-level, with each cell assumed to be independent of other cells. The failure rate of the device was then equally distributed over the cells. A more realistic approach, and a less conservative one from the standpoint of coverage, would have been to emulate the buffers, column and row decoders, as well as the memory cells. From "MIL-HDBK-217C, Notice 1, 72^{o}C, uninhibited aircraft environment" the proportion of failure rate for these components (54S5472) is

0.326 for the memory cells
0.674 for the buffer and decoders, etc.

If it assumed that detection of faults in the buffers and decoders is 100% then the 114 undetected faults is Partition 5 (of Table 13) become

$$0.326 \times 114 = 37$$

and the resultant number of detected faults would be

219 instead of 142.

The resultant coverage would then be

$$\frac{2409}{2534} = 0.951 \quad (95.1\%).$$

7.3.1 Indistinguishable Faults

Based on a sample of 6,600 gate-level faults the proportion of indistinguishable faults are given in Table 17, by partitions. The large proportion of indistinguishable faults (i.e., 16.5%) and the necessity to identify each one in the estimation of coverage were the major obstacles in the self-test design and validation process. We note that the proportion of indistinguishable faults was estimated to be 23.7% in the earlier study, based on a sample of 300 gate-level faults.

7.4 Summary and Conclusions

- Emulation appears to be an indispensable tool in the design and validation of an efficient self-test program.
- Self-test should be designed to capitalize on the hardware mechanization of the instruction set. Emphasis should be placed on detecting faults in the least reliable component.
- Pin-level faults are easier to detect than gate-level faults. The latter tend to be highly data-dependent. Coverage of pin-level faults did not change significantly between the initial self-test and the final revision.

- The "box" score for the initial and final self-test program is

	Coverage			
	Gate-Level	Component-Level	Words of Memory	Cycle Time
Initial	86.5%	97.9%	1100	7777
Final	92.0%	97.6%	346	2062

- The worst coverage was in the micromemory. A more realistic emulation of these devices, which included buffers, column and row decoders, would have yielded a significant improvement in total coverage, e.g., to 95.1%,

- Based on our experience and observations, thus far, we conjecture that virtually any self-test program of 200 words or more, in which even a modest effort was taken to exercise the major hardware components, will yield a gate-level fault coverage of 85%.

  Coverages greater than 90%, however, are a different matter, as evidenced by the successive self-tests of Table 12. The situation could be improved significantly if processors incorporated a direct means of testing the micromemory and control proms either through a parity checker or,more preferably, by making the contents of the memories directly accessible to the programmer.

TABLE 12

CPU SELF-TEST REVISIONS
GATE-LEVEL COVERAGE

RUN	REVISION	WORDS	MICROCYCLES	FAULTS	$1 - \alpha$
1	0 (Initial)	1100	7777	300	86.4(%)
2	1	192	974	300	88.0
3	2	202	1017	300	88.6
4	3	230	1430	300	87.9
5	4	260	1698	300	92.3
6	5	295	1802	500	93.3
7	6	330	1992	1000	94.4
8	7	334	2043	1000	93.4
9	8 (Final)	346	2062	1000	91.9
10	8 (Final)	346	2062	1000	92.5
11	8 (Final)	346	2062	1000	91.5

$1 - \alpha$ = coverage when indistinguishable faults are disqualified

TABLE 13

FINAL SELF-TEST

GATE-LEVEL COVERAGE BY PARTITIONS

PARTITION	FAULTS INJECTED	NOT DETECTED DIST.	NOT DETECTED INDIST.	DETECTED	$1 - \alpha$
1	553	22	30	501	95.8(%)
2	475	12	37	426	97.3
3	639	11	75	553	98.0
4	741	14	32	695	98.0
5	501	114	245	142	55.5
6	91	29	47	15	34.1
	3000	202	466	2332	

$$1 - \alpha = \frac{2332}{2534} = .92 \quad (92.\%)$$

TABLE 14

INITIAL SELF-TEST

GATE-LEVEL COVERAGE BY PARTITIONS

PARTITION	TOTAL INJECTED	UNDETECTED DIST.	UNDETECTED INDIST.	DETECTED
1	34	4	2	28
2	74	7	9	58
3	55	0	16	39
4	74	9	6	59
5	50	8	33	9
6	13	3	5	5
	300	31	71	198

$$1 - \alpha = \frac{198}{229} = .865 \quad (86.5\%)$$

TABLE 15

FINAL SELF-TEST

COMPONENT-LEVEL COVERAGE BY PARTITIONS

PARTITION	FAULTS INJECTED	NOT DETECTED DIST.	NOT DETECTED INDIST.	DETECTED	$1 - \alpha$
1	76	3	1	72	96.0(%)
2	100	1	9	90	99.0
3	106	0	14	92	100
4	118	5	0	113	95.8
	400	9	24	367	

$$1 - \alpha = \frac{367}{376} = .976 \quad (97.6\%)$$

TABLE 16

INITIAL SELF-TEST

COMPONENT-LEVEL COVERAGE BY PARTITIONS

PARTITION	FAULTS INJECTED	NOT DETECTED	DETECTED
1	35	1	34
2	73	1	72
3	43	2	41
4	38	0	38
	189*	4	185

$$1 - \alpha = \frac{185}{189} = .979 \quad (97.9\%)$$

* 11 faults were disqualified as indistinguishable.

TABLE 17

PROPORTION OF INDISTINGUISHABLE FAULTS (γ)

PARTITION	TOTAL # INJECTED FAULTS	INDISTINGUISHABLE	γ
1	1174	57	.049
2	1150	88	.077
3	1352	186	.138
4	1633	72	.044
5	1100	589	.535
6	191	96	.503

TOTAL PROPORTION OF INDISTINGUISHABLE FAULTS = 16.5%

8.0 URN MODEL

8.1 Urn Model Description

Several models have been investigated in an attempt to characterize the dynamics of fault propagation in a digital computer. Although simplistic in their assumptions, these models may, nevertheless, provide insight into this undoubtedly complex process. It has been conjectured (ref. 2) that the distribution of latency can be modelled by analogy with balls in an urn. We prefer to employ a different analogy although the resultant distributions are the same.

We postulate that the computer can be subdivided into three sets of mutually exclusive components C_1, C_2, C_3 such that

C_1 = Set of components randomly exercised by the program

C_2 = Set of components continually exercised by the program

C_3 = Set of components never exercised by the program.

We make the further assumption that a fault is detected if and only if the faulted component is exercised. The scenario is that of an avionics computer executing two software programs one of which is executed full-time and the other, part-time. The components that are exercised by the full-time mode are denoted by C_2 and those exercised by the part-time mode by C_1. Neither the full-time or part-time modes exercise components, C_3.

We assume that the part-time mode is exercised randomly. If the unit of time is a repetition of the full-time program then we postulate that the excitation is poisson-distributed in time with a = probability that the part-time mode is exercised in a repetition of the full-time program.

Let λ_1 = Failure rate of C_1 (Failures/hour)

λ_2 = Failure rate of C_2 (Failures/hour)

λ_3 = Failure rate of C_3 (Failures/hour)

$\lambda = \lambda_1 = \lambda_2 + \lambda_3$ (Failures/hour)

We now derive the latency distribution given that a fault has just occurred. The distribution is defined in terms of three parameters, a, P and Q_0 where

P = Probability that the fault is detected in the first repetition given that it occurred in sets C_1 or C_2

Q_o = Probability that the fault is never detected.

It is easy to derive the following relationships:

1) $P_o = 1 - Q_o = \frac{\lambda_1}{\lambda} + \frac{\lambda_2}{\lambda}, \; Q_o = \frac{\lambda_3}{\lambda}$

2) $$P = \frac{\frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda}}{\frac{\lambda_2}{\lambda} + \frac{\lambda_1}{\lambda}} = \frac{\frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda}}{P_o},$$

where P_o = Probability that the fault is detected eventually.

If P_k = probability that the fault is detected in the k-th repetition and not detected in a previous repetition, $k = 1, 2, 3, \ldots, n$,

q_{n+1} = probability that the fault is not detected in the previous n repetitions,

then

$$P_1 = P_o\,P = \frac{\lambda_2}{\lambda} + a\,\frac{\lambda_1}{\lambda}$$

$$P_2 = (1 - P)\,a\,P_o = a\,(1 - a)\,\frac{\lambda_1}{\lambda}$$

3) $$\vdots$$

$$P_n = (1 - P)\,(1 - a)^{n-2}\,a\,P_o = a(1 - a)^{n-1}\,\frac{\lambda_1}{\lambda}, \; n = 2, 3, \ldots$$

$$q_{n+1} = Q_o + \sum_{k = n+1}^{\infty} P_k = Q_o + (1 - P)\,P_o\,(1 - a)^{n-1}$$

$$= \frac{\lambda_3}{\lambda} + (1 - a)^n\,\frac{\lambda_1}{\lambda}, \; n = 1, 2, 3, \ldots$$

Observe that

$$q_{n+1} + \sum_{k=1}^{n} p_k = 1, \text{ as expected.}$$

In estimating the above distribution the number of repetitions will be limited to eight. Then, the study will estimate the quantities

$$p_1, p_2, \ldots, p_8, q_9.$$

for S-a-1, S-a-0 and combined faults.

It is easy to shown that the Urn Model can be represented as a Markov Model, as shown in Figure 9.

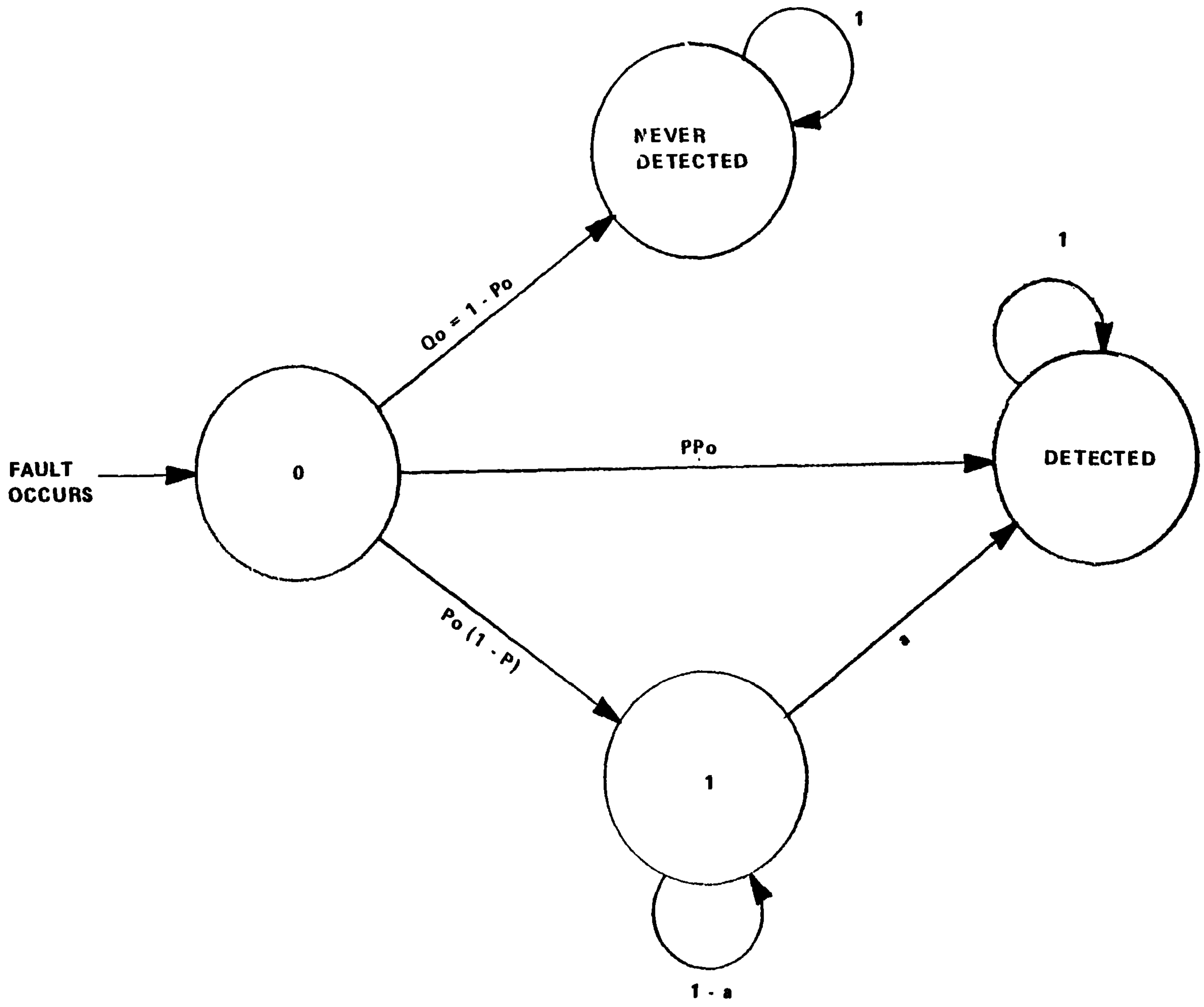


FIGURE 9 MARKOV MODEL REPRESENTATION OF THE URN MODEL

9.0 ESTIMATORS

As indicated previously, the principal objective of the study is to obtain estimates of fault coverage and fault latency in a typical avionics miniprocessor. Although the statistical experiments were carefully designed to yield high accuracy and confidence for the least cost the estimates should not be taken too literally. The reader is advised to exercise engineering judgement in interpreting the results especially when inferring conclusions that depend upon small differences in the estimates. The reason for caution is the uncertainty in the assumptions underlying the study - assumptions which may, if incorrect or inaccurate, contribute a far greater uncertainty to the results than the statistical analysis would imply.

For the record, the critical assumptions of the study are:

- From the standpoint of failure modes and effects every device can be represented by the manufacturer-supplied gate-level, equivalent circuit.
- Every fault can be represented as either a S-a-0 or S-a-1 at a gate node.
- The failure rate of each device is equally distributed over the gates of the gate-level equivalent circuit.
- The failure rate of each gate is equally distributed over the nodes of the gate.
- Memory failures are exclusively faults of single bits.

9.1 Estimators for Self-Test Coverage

The estimators for x, y and z are

1) $x^* = \frac{m_d}{m}$

2) $y^* = \frac{n_d}{n}$

3) $z^* = \frac{m_d + n_d}{m + n}$

where

x, y, z = probability that a S-a-0, S-a-1, combined fault is detected;

m_d, n_d = number of S-a-0, S-a-1 faults detected;

m, n = number of S-a-0, S-a-1 faults injected.

A more accurate estimate of z can be obtained if stratified sampling is employed. For example, let

a_x = proportion of S-a-0 faults in the fault set of the processor

a_y = proportion of S-a-1 faults in the fault set of the processor

where $a_x + a_y = 1.$

If m and n are selected such that

$$m = a_x N, \; n = a_y N$$

where

N = total number of faults injected,

then

$$z^* = a_x x^* + a_y y^*$$

is more accurate than (3) if $x \neq y$. Although stratified sampling was not intentionally employed in the study the actual selection resulted in an almost equal number of S-a-0 and S-a-1 faults.(*)

9.2 Estimators for Latency

The estimators for x_k, y_k an z_k are

$$x_k^* = \frac{m_k}{n}$$

4) $$y_k^* = \frac{n_k}{n}$$

$$z_k^* = \frac{m_k + n_k}{m + n} \; , \; k + 1, 2, 3, \ldots, 8,$$

* In the selection process $a_x = a_y = 0.5$, i.e., S-a-0 and S-a-1 faults were equally likely.

where

x_k, y_k, z_k = probability that a S-a-0, S-a-1, combined fault is detected in the k-th repetition;

m_k, n_k = number of S-a-0, S-a-1 faults detected in the k-th repetition.

With some abuse of terminology we define

x_9, y_9, z_9 = probability that a S-a-0, S-a-1, combined fault is <u>not</u> detected in the previous 8 repetitions.

We note that x_9 corresponds to q_9 of Section 8. The estimators for x_9, y_9 and z_9 are

$$x_9^* = \frac{m - m_1 - m_2 - \dots - m_8}{m} = 1 - x_1^* - x_2^* - \dots - x_8^*$$

5) $$y_9^* = \frac{n - n_1 - n_2 - \dots - n_8}{n} = 1 - y_1^* - y_2^* - \dots - y_8^*$$

$$z_9^* = \frac{m\, x_9^* + n\, y_9^*}{m + n} = 1 - z_1^* - z_2^* - \dots - z_8^*.$$

9.3 Estimators for Urn Model Parameters

The method of estimation will be described for S-a-0 latency distributions. With an obvious change in parameters, e.g., m_k, the estimates can be applied to S-a-1 and combined latency distributions, as well.

The method is based on the principal of maximum likelihood. We note that m_k S-a-0 faults are detected in the k-th repetition. Accordingly, we seek Urn Model parameters a, P and P_0 that maximize the likelihood function

$$L = p_1^{m_1}\, p_2^{m_2} \dots p_8^{m_8}\, q_9^{m_9}$$

where

$$p_1 = P_o P$$

$$p_2 = (1 - P)\ a\ P_o$$

$$p_3 = (1 - P)\ a\ P_o\ (1 - a)$$

6) $$\vdots$$

$$p_8 = (1 - P)\ a\ P_o\ (1 - a)^6$$

$$q = Q_o + (1 - P)\ P_o\ (1 - a)^7$$

and $m_9 = m - m_1 - m_2 - \dots - m_8$

(See Section 8.1 for a definition of the Urn Model).

The maximum likelihood estimators for a, P and P_o are obtained as the solution of

$$\frac{\partial L}{\partial a} = 0,\ \frac{\partial L}{\partial P} = 0,\ \frac{\partial l}{\partial P_o} = 0.$$

Instead of solving these equations for the maximum likelihood estimators, we will employ an approximation that was suggested in (ref. 2). There, it was assumed that

$$q_9 = 1 - P_o = Q_o.$$

In other words, detectable faults are always detected in the first 8 repetitions. From (6) this is equivalent to the approximation

7) $$(1 - P)\ P_o\ (1 - a)^7 = 0.$$

If this substitution is made in the likelihood function, L, then the resultant estimates are, for S-a-O faults,

$$p_o^* = \frac{1}{m} \sum_{i=1}^{8} m_i$$

$$p^* = \frac{m_1}{\sum_{i=1}^{8} m_i}$$

8)

$$a^* = \frac{\sum_{i=1}^{8} m_i - m_1}{\sum_{i=1}^{8} i m_i - \sum_{i=1}^{8} m_i}$$

The results of (ref. 1) confirmed the accuracy of these approximations.

9.4 Accuracy and Confidence of Coverage Estimates

9.4.1 Self-Test Coverage

It can be shown (ref. 3) that

9) $E\ (x^*) = x,\ E\ (y^*) = y,\ E\ (z^*) = z$

and

$$E\ (\ (x - x^*)^2\ ) = \frac{x\ (1 - x)}{m}$$

10) $$E\ (\ (y - y^*)^2\ ) = \frac{y\ (1 - y)}{n}$$

$$E\ (\ (z - z^*)^2\ ) = \frac{z\ (1 - z)}{N}$$

where

$E\ (\cdot) =$ expected value of $(\cdot)$.

For m, n and N sufficiently large the estimators x*, y* and z* are approximately Gaussian with means and variances given by (9) and (10), respectively.

The following derivation of accuracy and confidence is general and applies to any quantity, x, estimated by the method of Section 9.1. As before,

x^* = estimate of x

m = sample size.

It is well-known (See (ref. 4), for example) that the probability that x lies between the limits

$$\frac{m}{m+\lambda^2}\left(x^* + \frac{\lambda^2}{2m} \pm \lambda\sqrt{\frac{x^*(1-x^*)}{m} + \frac{\lambda^2}{4m^2}}\right)$$

or, equivalently, that x* lies between the limits

11) $x \pm \lambda\sqrt{\frac{x(1-x)}{m}}$

is equal to γ, where γ is the area of the standard Gaussian distribution between $-\lambda$ and λ. From (10) we may say that the error in the estimate, x*, is

12) $\varepsilon = \lambda\sqrt{\frac{x(1-x)}{m}}$

with a confidence level of γ.

Equation (12) is an ellipse in x and ϵ. Table 18 gives a tabulation of $\varepsilon\sqrt{m}$ versus x for a confidence level of γ = .95.

It is often convenient to obtain error estimates that are independent of x. From (12) it can be seen that the maximum error occurs when x = ½. Table 19 gives a tabulation of this maximum error versus sample size and confidence level. It is noted that the maximum error can be extremely conservative.

9.4.2 Latency Estimate

For the latency distributions the estimate of most interest is the coverage after 8 repetitions. The accuracy and confidence of these estimates are obtained exactly as for self-test coverage estimates. Thus, if

z^* = estimated coverage of combined faults after 8 repetitions, then

$$E\left((z - z^*)^2 \right) = \frac{z\,(1 - z)}{m}\ .$$

9.4.3 Urn Model Parameter Estimates

It was shown in (ref. 1) that, using the estimators of (8), we obtain

$$E\left((P - P^*)^2 \right) = \frac{P\,(1 - P)}{m\,P_o}$$

$$E\left((P_o - P_o^*)^2 \right) = \frac{P_o\,(1 - P_o)}{m}$$

$$E\left((a - a^*)^2 \right) = \frac{a^2\,(1 - a)}{m\,P_o\,(1 - P)}$$

and the cross-covariances vanish. Thus the estimates are independent and, at a confidence level of γ, the errors are, for P, P_o, a, respectively,

$$\varepsilon_p = \lambda\sqrt{\frac{P\,(1 - P)}{m\,P_o}}$$

$$\varepsilon_{P_o} = \lambda\sqrt{\frac{P_o\,(1 - P_o)}{m}}$$

$$\varepsilon_a = \lambda\sqrt{\frac{a^2\,(1 - a)}{m\,P_o\,(1 - p)}}$$

where λ is as defined in Section 9.4.1.

TABLE 18

ERROR FOR A CONFIDENCE LEVEL OF $\gamma = .95$

$$\varepsilon\sqrt{m} = \lambda\sqrt{x(1-x)}$$

$\varepsilon\sqrt{m}$	x
0.0	0
.427	.05
.588	.1
.70	.15
.784	.2
.849	.25
.898	.3
.935	.35
.960	.4
.975	.45
.98	.5
.975	.55
.96	.6
.935	.65
.898	.7
.849	.75
.784	.8
.7	.85
.588	.9
.427	.95
0.0	1.0

TABLE 19

WORST CASE GAUSSIAN
ERROR VERSUS
SAMPLE SIZE AND CONFIDENCE LEVEL

CONFIDENCE LEVEL \ SAMPLE SIZE	200	300	400	600	1000
.6	.03	.025	.021	.017	.013
.7	.037	.03	.026	.021	.017
.8	.046	.038	.033	.027	.021
.9	.058	.048	.041	.034	.026
.95	.069	.056	.049	.04	.031

10.0 EMULATION CHARACTERISTICS

10.1 BDX-930 Architecture

The BDX-930 Digital Processor is a microprogrammed, pipelined machine designed around the AMD2901A four bit microprocessor slice. The machine contains sixteen general purpose registers of which four registers may be loaded directly from memory and two registers may be used as base registers. One register is used as a stack pointer.

The program counter and memory address register are contained in the 9407, a chip designed to perform memory address arithmetic. Along with a temporary register contained on the same chip, the BDX-930 is able to perform four basic addressing modes involving three registers and various instruction fields.

The machine contains three memory interface data registers which are used to input and output memory data. There are also a number of one bit status flag registers that can be manipulated under program control. This includes the F1 and F2 registers, which are hardware flags, and the interrupt enable, overflow status registers. There also exist the indirect and link registers used by the microcode for branching.

The microcode is contained in seven proms and a pipeline register is included for simultaneous microcode fetch and decoding. Various internal and external conditions can affect microcode branching as selected by the microcode itself and a microcode control prom. In addition to a rich instruction set which includes 16 and 32 bit fixed point operations, there is a test set interface in the microcode. A selectable saturate mode is available which limits the results of arithmetic operations when overflow or underflow occur.

For simulation purposes, the computer has been divided into six partitions, consisting of the following principal devices:

Partition 1 - Address Processor

- 4 - 9407 Memory Address Processor Equivalent Circuit
- Selector Chips to Multiplex Memory Address Source

  .. 4 - 54LS352 4:1
  .. 2 - 54LS158 2:1

Partition 2 - Data and Status Registers

- 2 - 54LS374 Memory Input Buffer Register
- 2 - 54LS374 Memory Output Buffer Register

- 2 - 54LS374 Next Instruction Register
- 3 - 54LS113 Single Bit Registers for
  - .. overflow
  - .. indirect addressing
  - .. link (bit carry for divide)
  - .. interrupt mode
  - .. F1 and F2
- 2 - 54LS153 Select Overflow, Link, and Indirect Bit Sources
- 2 - 54LS245 Octal Bus Transceivers

Partition 3 - Microcontroller

- Pipeline Register
  - .. 4 - 54LS273 Octal Latch
  - .. 4 - 54LS175 Quad Latch
  - .. 1 - 54LS374 Octal Latch With Tri-State
- 1 - 54LS273 External Signal Synchronizer
- 3 - 54LS151 Selectors 8:1 for Branch Conditions
- 1 - 54LS169 Counter for Shift and Multiply Instructions
- 1 - 54LS169 Counter for Multiple Register Load-Store Instructions
- 1 - 54LS377 Instruction Register
- 1 - 54LS253 Microcode Branch Selector

Partition 4 - Execute

- 4 - AMD2901A 4 Bit Slice ALU
- 1 - AMD2902 Lookahead Carry
- 2 - 54LS153 Selector 4:1 Register Selectors
- 1 - 54LS253 Selector 4:1 Shift Bit Selector

Partition 5 - Microcode

- 7 - 54S472 Proms with 56 Bit Wide Microcode

Partition 6 - Control Proms

- 1 - 54S472 Prom Microcode Start Address for Macroinstructions
- 1 - 54S288 Prom Control for Microcode Branch

Instruction execution is accomplished by a pipelined architecture; various stages of execution occur simultaneously for a sequence of instructions. Consider, for instance, four instructions, A,B,C,D, to be executed in sequence. During the same clock cycle it is possible for the program counter to be incremented to point to instruction D, while instruction C is being fetched, instruction B is being decoded and instruction A is being executed.

With this level of parallelism, it will be noted that when the execution phase of an instruction is one clock cycle, the average time to perform the entire instruction will be one clock cycle.

It should also be noted that the partitioning of the BDX-930 is roughly broken up into the stages of the pipe: - address, fetch, decode, and execute. These stages of the pipe are joined by various buses throughout the CPU. These buses are formed from tri-state logic and some are bidirectional. An enumeration of the major buses includes

- Y - Connects the output of the ALU (AMD2901A) to the address processor and the output register. In addition, it connects the output of the next instruction buffer to the start address register and instruction register.
- D - Connects the memory data register and the program counter to the input of the ALU.
- DAT - Bidirectional bus connecting memory and I/O to the memory data register and output register.
- M - Bidirectional memory data bus
- MAR - Memory Address Bus
- U - Microcode Bus
- IR - Instruction Register

A list of the devices used in the BDX-930 and their failure rates is given in Table 20. The data was obtained from MIL-HDBK217B Notice 2.

10.2 Description of the Emulator

The emulation includes the components of the CPU (Central Processor Unit), scratchpad memory and those portions of the program memory containing the target programs and the target self-test program. The emulation is derived from the circuit schematics. Each device is represented by a gate-level equivalent circuit supplied by the chip manufacturer. It was found that six types of gates were sufficient to represent any device, e.g., NAND, AND, OR, NOT, NOR, EXCLUSIVE OR. Table 21 gives the number of equivalent gates in each device of the CPU. In all, 5,100 gates were required. In the interests of reducing execution time, it was not expedient to emulate all components at the gate-level. The following elements are represented at the functional-level:

program memory
scratchpad memory
microprogram and control memories
16 general purpose arithmetic registers.

The emulation did not include the direct memory access unit (DMA) or any of the devices of the I/O. The emulated devices of the CPU are shown in Figure 10.

Faults were injected into all devices except the program and scratchpad memories. Because the program memory is "read-only", no processor, faulted or not, is permitted to write into this memory. However, even though the scratchpad memory is never faulted, a faulty processor can write into it. As a consequence, in the parallel mode of operation where 32 processors are simultaneously emulated, the corresponding 32 scratchpad memories are also emulated.

No delay has been simulated between logic gates. It is assumed that all combinatorial logic is stable at the output the instant an input pattern is applied to it. This means that each time the input is changed, the network need only be evaluated once to supply the correct output pattern. Operating in this manner is very time efficient, but puts stringent requirements on the order of evaluation of the gates. To be able to meet these requirements, the logic is levelized, i.e., placed in groups or levels that represent the proper order of evaluation.

The emulator utilizes the parallel method of logic simulation and was hosted on a VAX-11/780. The data word of a VAX-11 contains 32 bits; each bit position is used to represent a different machine. The simplest gate operations are represented by a single Boolean instruction; when the two inputs occupy the same bit positions in their respective words, the output also occupies this bit position. The advantage of this technique is execution time savings. Typically, the amount of code necessary to simulate 32 machines is of the same order as the amount of code necessary to simulate only one machine. The BDX-930 description is contained in compiled code, rather than in tables, which was also done for speed.

Certain portions of the machine, notably the memory elements, were represented at a functional level rather than a gate level. For microprogram memory, two words of VAX-11 storage contain 56 bits of microstore; at micro memory fetch time, these bits are retrieved from the proper address for each of the simulated machines and combined to form suitable words to interface the gate portion of the emulation. The ROM portion of main memory is handled in the same manner. Writable store contains a routine to translate the gate inputs into consecutive VAX-11 storage words so that there is one copy of writable storage for each machine being emulated. On reading this storage, the process is reversed.

In a typical run of the emulator, 32 different machines are exercised; 31 faulted machines and one good machine. Each faulted machine is assumed to have a single hard fault at one node, either stuck-at-one (S-a-1) or stuck-at-zero (S-a-0). The faults are injected by defining extra gates at each node, an AND gate for stuck at zero and an OR gate for stuck-at-one. A typical AND gate using this technique is shown in Figure 11.

When the entire emulation is executed for true values, the ratio of VAX-11 time to BDX-930 time is 5000:1; with faults injected in one partition, the number is 7000:1.

TABLE 20

COMPONENTS OF THE BDX-930 CPU

DEVICE	FAILURE RATE/PER UNIT (PPMH)
9407	1.3931
2901A	2.1656
2902	0.3898
5440	0.0653
54125	0.0855
54S00	0.0855
54S04	0.1003
54S10	0.0764
54S20	0.0654
54S32	0.2138
54S288 (32x8 prom)	0.1787
54S472 (512x8 proms)	1.008
54LS00	0.084
54LS02	0.084
54LS04	0.0983
54LS08	
54LS11	0.0752
	0.084
54LS86	0.084
54LS113	0.1447
54LS151	0.1483
54LS153	0.1447
54LS158	0.1410
54LS169	0.6603
54LS175	0.1703
54LS245	0.3792
54LS253	0.1447
54LS273	0.6882
54LS352	0.3117
54LS367	0.1100
54LS374	0.7234
54LS377	0.7148

TABLE 21

MICROCIRCUITS AND EQUIVALENT GATE COUNT

DEVICE	EQUIVALENT GATES
2901A	798
2902	19
54113	8
54151	17
54153	16
54158	15
54169	58
54175	22
54245	18
54253	16
54273	34
54352	16
54374	26
54377	35
9407	143

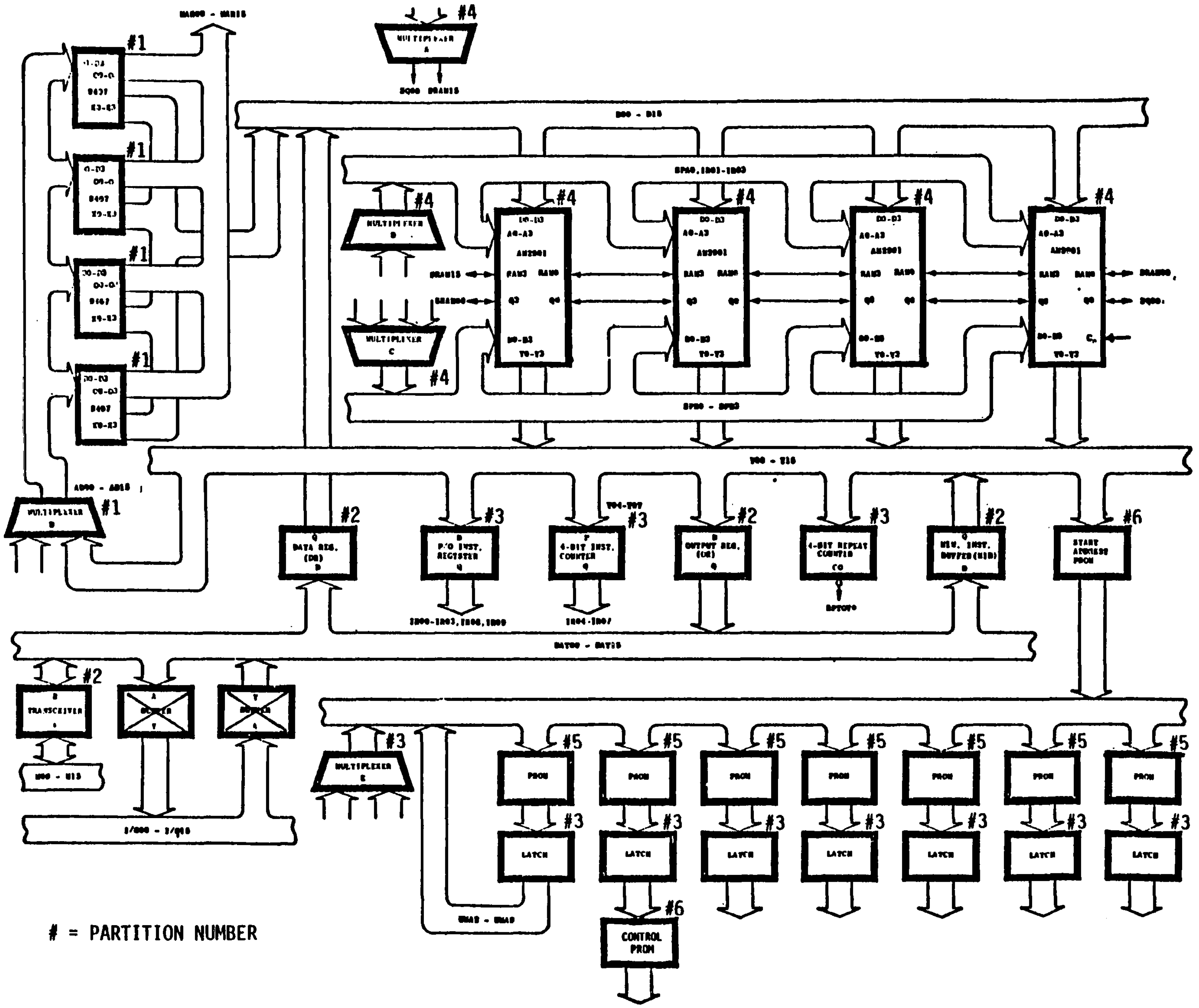


FIGURE 10 BDX-930 PROCESSOR

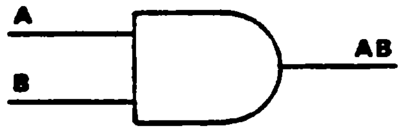


FIGURE 11A
NON-FAULTED "AND" GATE

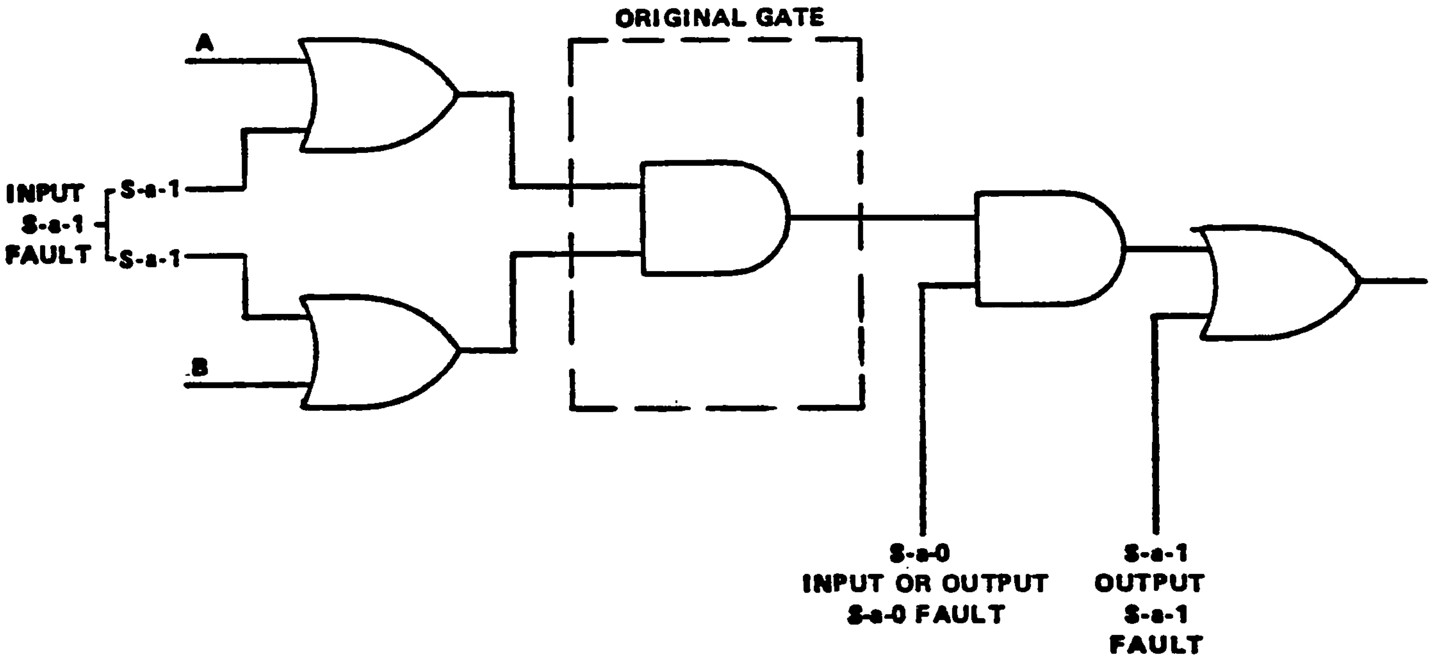


FIGURE 11B

FAULT MODEL OF "AND" GATE

11.0 CONCLUSIONS

On the basis of the study we conclude:

- The present study substantiates the results of the previous study. The only difference was in the conjecture that detection is a linear function of the number of instructions. The present study demonstrates that coverage is independent of the length of the program.
- Emulation is a practicable approach to failure modes and effects analysis of a digital processor.
- The run time of the emulated processor on a VAX-11/780 host computer is only 5000 to 7000 times slower than the actual processor. As a consequence, large numbers of faults can be studied at relatively little cost and in a timely manner.
- The fault model, although somewhat arbitrary, can be updated as more data becomes available.
- Gate-level faults are more difficult to detect than component-level faults. As a consequence, coverage requirements should be explicit as to the types of faults to be covered.
- In a comparison-monitored system the accumulation of latent faults can be significant. For example, in a flight control program of 2200 instruction, 21% of all distinguishable faults remained undetected after 8 repetitions. The impact of this accumulation on aircraft survivability has yet to be determined.
- Self-test should be designed to capitalize on the hardware mechanization of the CPU.
- It is relatively easy to generate a self-test with a gate-level coverage between 85% and 90%, To obtain a coverage of 95% is difficult.
- It is relatively easy to obtain component-level coverage in excess of 95%.
- Faults in the micromemory are difficult to detect. This situation could be improved if future processors incorporated a direct means of testing, either by a parity check or, more preferably, by making the contents of the micromemory accessible to the programmer.
- A large proportion of faults, i.e., 16.5% were indistinguishable (i.e., "don't care"). It was extremely difficult to identify these faults.

- The Urn Model can characterize the shape of the latency distribution. This can be attributed to:
  1) The monotonic, decreasing property of the empirical distribution
  2) The 3 degrees-of-freedom which the model provides for a best fit.
- It is doubtful that the Urn Model parameters can be predicted for a program on the basis of length or instruction mix.

12.0 REFERENCES

1. McGough, J., Swern, F., "Measurement of Fault Latency in a Digital Avionic Mini Processor", NASA CR-3462, NASA Langley Research Center, Hampton, VA, October, 1981.

2. Nagel, P., "Modeling of a Latent Fault Detector in a Digital System", NASA CR-145371, 1978.

3. McFarlane Mood, A., "Introduction to the Theory of Statistics", McGraw-Hill; New York, 1950.

4. Cramer, H., "Mathematical Methods of Statistics", Princeton University Press; Princeton, 1958.

1. Report No. NASA CR-3651	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle MEASUREMENT OF FAULT LATENCY IN A DIGITAL AVIONIC MINI PROCESSOR - PART II		5. Report Date January 1983
		6. Performing Organization Code
7. Author(s) John G. McGough and Fred Swern		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address Flight Systems Division Bendix Corporation Teterboro, N.J. 07608		11. Contract or Grant No. NAS1-15946
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley NASA Project Engineer: Salvatore J. Bavuso
Progress Report

16. Abstract

This report describes the results of fault injection experiments utilizing a gate-level emulation of the central processor unit of the Bendix BDX-930 digital computer. The study is an extension of a previous study:

Measurement of Fault Latency in a Digital Avionic Mini Processor
NASA CR-3462, October 1981.

The poor coverage of comparison-monitoring, which the earlier study demonstrated, could have been due to the limited repertoire of the instruction set used. As a consequence, it was decided to reprogram several earlier programs but this time expanding the instruction set to capitalize on the full power of the BDX-930 computer. As a final demonstration of fault coverage an extensive, 3-axis, high performance flight control computation was added.

A secondary objective of the study was to demonstrate the stages in the development of a CPU self-test program emphasizing the relationship between fault coverage, speed and quantity of instructions.

17. Key Words (Suggested by Author(s)) Emulation, Gate-Level, Fault Detection, Fault Latency, Comparison-Monitoring, Self-Test	18. Distribution Statement Unclassified - Unlimited Subject Category 59

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	94	A05